Leveraging static dataflow in dynamically-compiled pipeline-parallel text-parsing programs using the Parabix framework

by

Nigel W. Medforth

M.Sc., Simon Fraser University, 2013
B.Tech., Kwantlen Polytechnic University, 2009

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
School of Computing Science
Faculty of Applied Sciences

© Nigel W. Medforth 2022
SIMON FRASER UNIVERSITY
Summer 2022

Copyright in this work is held by the author. Please ensure that any reproduction
or re-use is done in accordance with the relevant national copyright legislation.

Declaration of Committee

Name:	**Nigel W. Medforth**
Degree:	**Doctor of Philosophy**
Title:	**Leveraging static dataflow in dynamically-compiled pipeline-parallel text-parsing programs using the Parabix framework**
Committee:	**Chair:** Yuepeng Wang Assistant Professor, Computing Science
	Robert Cameron Co-Supervisor Professor, Computing Science
	Nick Sumner Co-Supervisor Associate Professor, Computing Science
	Thomas Shermer Committee Member Professor, Computing Science
	Tianzheng Wang Examiner Assistant Professor, Computing Science
	Daniel Lemire External Examiner Professor, Science and Technology Université du Québec (TELUQ)

Abstract

Modern Big Data systems have shifted away from pure ETL solutions and towards real-time analysis of "raw text". Unsurprisingly, efficient analysis of raw text is a critical performance concern for Big Data systems. Parabix is an intrinsically parallel programming paradigm that is well suited for high-speed text matching, validation and transcoding problems. This framework combines the inherit SIMD bit-parallelism of the `Pablo` programming language, the thread-parallelism of linear-pipeline programming, and the dynamic-compilation capabilities of LLVM `MCJIT` to create ad-hoc text-query applications.

In this dissertation, we extend the existing Parabix framework with a more generalized notion of processing rates, I/O attributes and automate the memory management of the program. Our goal is to support a wider range of programs than prior frameworks allowed and to reduce programmer effort when constructing highly-parallel programs. To do so, we incorporate dataflow-programming techniques, which allow us to better analyze pipeline stage relationships and automatically compile efficient linear-pipelines for variable-rate systems. We extend upon existing literature to provide a scheduling algorithm tailored for such systems. Our goal is reliably provide a near-linear multi-threaded speedup on commodity hardware for up to $\lfloor \texttt{total single-threaded time} / \max(\texttt{individual pipeline stage time}) \rfloor$ threads.

Compared to the prior version of the Parabix framework, our work increases the average program acceleration by 8.2 – 63.9%, with the best improvements occurring on newer architectures with a greater number of threads. Our evaluation shows that by combining our fixed-data linear-pipeline model, data-parallel execution of state-free stages, and dataflow programming concepts, we can exceed our expected ideal speed-up limit for two of the ten test cases by nearly 40% and 60%, achieving a 3.5× acceleration with four threads. For six of the remaining eight cases, the framework attains at least 83% of the limit, with half being over 90%. We assess each program in detail — including the final problematic test case that only reaches 71% of our goal — and provide recommendations for future work.

Keywords: Big data; data parallelism; dataflow programming; dynamic compilation; linear-pipeline parallelism; stream processing

Dedication

This dissertation is dedicated to my parents, Claire and Henry Medforth, who raised me to be curious, nurtured my interest in Computer Science from an early age and supported me on every step of this journey.

Acknowledgements

Large endeavours cannot be completed in isolation. I would not have been able to develop this framework without the support of my supervisors and lab mates. For that, I'm very grateful to Robert Cameron, Nick Sumner and Thomas Shermer for their advice over the years as well as Kenneth Herdy and Dan Lin who helped teach me what Pablo and Parabix were during the early stages and whose camaraderie has been greatly missed. I'd also like to thank my friends, Jeffery Jonsson and Kymberly Holt, who encouraged me and helped keep me sane during this process, Michael Bendner, who was the first to show me the fun side of programming, and my family, who enabled me to complete this dissertation. Last but not least, I would to thank the technical staff at SFU, especially Jason Ashby, who was never slow in responding to a request, my instructors at KPU, Andy Law and Mehdi Talwerdi, who encouraged me to consider graduate studies, and the `boost`, `C++`, `LLVM` and `Z3` communities as a whole, whose works I relied and/or built upon.

Table of Contents

Declaration of Committee ii

Abstract iii

Dedication iv

Acknowledgements v

Table of Contents vi

List of Tables xi

List of Figures xii

Glossary xv

1 Introduction 1

1.1 Brief overview of the Parabix framework . . . 1
1.2 Dataflow models . . . 2
1.3 Contrasting traditional and Parabix dataflow models . . . 5
1.4 Research objective . . . 6
1.5 Limitations . . . 7
1.6 Contributions . . . 7
1.7 Thesis structure . . . 8

2 Description of the Parabix framework 9

2.1 High-level design . . . 10
2.1.1 Kernels . . . 11
2.1.2 Processing rates . . . 12
2.1.3 Streamsets . . . 16
2.1.4 Scalars . . . 18
2.1.5 Pipeline . . . 20
2.1.6 Attributes . . . 26
2.1.7 Object cache . . . 30

2.1.8 Kernel builder . . . 33
2.1.9 Kernel compiler . . . 38
2.1.10 Pipeline compiler . . . 38
2.1.11 JIT driver . . . 38
2.2 Common Parabix kernels . . . 39
2.2.1 I/O kernels . . . 39
2.2.2 Pablo-support kernels . . . 40
2.2.3 Filtering kernels . . . 42
2.2.4 Spreading kernels . . . 43
2.3 Overview of the Pablo architecture . . . 44
2.3.1 Pablo language . . . 45
2.3.2 BixNum . . . 63
2.3.3 Difference between the Pablo block and Pablo builder . . . 64
2.3.4 Pablo compiler and optimization passes . . . 64

3 Dataflow programming **70**
3.1 Synchronous dataflow . . . 70
3.1.1 Constructing a synchronous dataflow graph . . . 71
3.1.2 Generating synchronous dataflow schedules for Parabix programs . . . 74
3.2 Offline dynamic storage allocation for dataflow programs . . . 80
3.2.1 Weighted interval graph colouring . . . 80
3.2.2 Representing Parabix schedule generation as a DSA problem . . . 81
3.2.3 Yang et al.'s colouring algorithm . . . 82
3.3 Related works . . . 85

4 Pipeline compiler **86**
4.1 Conversion of program graph to internal graph . . . 87
4.2 Partitioning of Parabix programs into synchronous subgraphs . . . 87
4.2.1 Identifying partitions . . . 88
4.2.2 Preliminary SDF-partitioning pass . . . 89
4.2.3 Variable-rate simulator . . . 90
4.2.4 Final partitioning pass . . . 93
4.3 Schedule generation of partitioned program . . . 94
4.3.1 Intra-partition schedule generation . . . 94
4.3.2 Inter-partition schedule generation . . . 96
4.4 Thread-local memory layout . . . 100
4.4.1 Brief review of cache prefetching . . . 101
4.4.2 Prefetch-friendly memory assignment . . . 102
4.5 Hybrid pipeline model . . . 103

5 Evaluation **104**
5.1 Experimental platform and methodology . . . 104
5.1.1 Applications . . . 106
5.1.2 Datasets . . . 106
5.1.3 RE queries . . . 107
5.1.4 Platforms . . . 108
5.2 Average multi-thread speedup evaluation . . . 109
5.2.1 Case study: `base64` . . . 112
5.2.2 Case study: `csv2json` . . . 113
5.2.3 Case study: `editd` . . . 114
5.2.4 Case study: `gb18030` . . . 115
5.2.5 Case study: `icgrep -colours=never` . . . 116
5.2.6 Case study: `icgrep -colours=always` . . . 117
5.2.7 Case study: `u8u16` . . . 118
5.2.8 Case study: `u32u8` . . . 119
5.2.9 Case study: `ztf-hash compression` . . . 120
5.2.10 Case study: `ztf-hash decompression` . . . 121
5.3 Average platform speedup evaluation . . . 122
5.4 Comparison against 2017 version . . . 123
5.4.1 Case study: `base64` . . . 124
5.4.2 Case study: `editd` . . . 125
5.4.3 Case study: `u8u16` . . . 126
5.5 Comparison against 2020 version . . . 128
5.5.1 Case study: `icgrep -colours=never` . . . 130
5.6 Comparison against hybrid pipeline . . . 132
5.7 Comparison against non-Parabix applications . . . 134
5.7.1 Case study: `base64` vs. `fastbase64` . . . 135
5.7.2 Case study: `csv2json` vs. sequential `csv2json` . . . 135
5.7.3 Case study: `icgrep` vs. `ugrep` . . . 136
5.7.4 Case study: `u8to16` and `u32to8` vs. `utf8lut` . . . 138

6 Conclusion and future work **140**

Bibliography **142**

Appendix A Two modifications to Yang et al.'s offline DSA algorithm **151**
A.1 Evaluation . . . 153
A.1.1 Experiment 1. . . . 153
A.1.2 Experiment 2. . . . 155

Appendix B Closest-match EA repair function **163**

Appendix C Program dataflow graphs **168**

C.1 base64 . . . 169
C.2 csv2json . . . 170
C.3 editd . . . 171
C.4 gb18030 . . . 172
C.5 icgrep -colours=never . . . 173
C.6 icgrep -colours=always . . . 174
C.7 u8u16 . . . 175
C.8 u32u8 . . . 176
C.9 ztf-hash compression . . . 177
C.10 ztf-hash decompression . . . 178

Appendix D Detailed evaluation data tables **179**

D.1 Multi-thread speedup evaluation . . . 179
D.1.1 Case study: `base64` . . . 179
D.1.2 Case study: `csv2json` . . . 181
D.1.3 Case study: `editd` . . . 182
D.1.4 Case study: `gb18030` . . . 184
D.1.5 Case study: `icgrep -colours=never` . . . 185
D.1.6 Case study: `icgrep -colours=always` . . . 190
D.1.7 Case study: `u8u16` . . . 195
D.1.8 Case study: `u32u8` . . . 196
D.1.9 Case study: `ztf-hash compression` . . . 198
D.1.10 Case study: `ztf-hash decompression` . . . 199
D.2 Selected examples of strides per segment diagrams . . . 201
D.2.1 Case study: `icgrep -always` . . . 201
D.2.2 Case study: `gb18030` . . . 203
D.2.3 Case study: `u32u8` . . . 205
D.3 Comparison against 2017 version . . . 207
D.3.1 Case study: `base64` . . . 207
D.3.2 Case study: `editd` . . . 210
D.3.3 Case study: `u8u16` . . . 213
D.4 Comparison against 2020 version . . . 215
D.4.1 Case study: `base64` . . . 215
D.4.2 Case study: `csv2json` . . . 218
D.4.3 Case study: `editd` . . . 221
D.4.4 Case study: `gb18030` . . . 224
D.4.5 Case study: `u8u16` . . . 227

D.4.6 Case study: `u32u8` . . . 230
D.4.7 Case study: `icgrep -colours=never` . . . 233
D.4.8 Case study: `icgrep -colours=always` . . . 254
D.4.9 Case study: `ztf-hash compression` . . . 275
D.4.10 Case study: `ztf-hash decompression` . . . 278
D.5 Comparison against hybrid pipeline . . . 281
D.5.1 Case study: `icgrep -colours=never` . . . 281
D.6 Comparison against non-Parabix applications . . . 301
D.6.1 Case study: `base64` vs. `fastbase64` . . . 301
D.6.2 Case study: `csv2json` vs. sequential `csv2json` . . . 302
D.6.3 Case study: `icgrep` vs. `ugrep` . . . 303
D.6.4 Case study: `u8to16` vs. `utf8lut` . . . 309
D.6.5 Case study: `u32to8` vs. `utf8lut` . . . 309

List of Tables

Table 5.1 Primary UTF-8 text datasets . . . 107
Table 5.2 CSV datasets . . . 107
Table 5.3 Converted text datasets file sizes . . . 107
Table 5.4 REs used in `icgrep` evaluations . . . 108
Table 5.5 `SSE-4.2` . . . 108
Table 5.6 `AVX2` . . . 108
Table 5.7 `AVX-512` . . . 109
Table 5.8 `base64` PAPI normalized difference comparison between `2017` and `CURR` 125
Table 5.9 `editd` PAPI normalized difference comparison between `2017` and `CURR` 126
Table 5.10 `u8u16` PAPI normalized difference comparison between `2017` and `CURR′` 127
Table 5.11 Average relative cost (%) of most expensive `u8u16` kernel for `CURR′` . . 127
Table 5.12 `icgrep` PAPI normalized difference comparison between `2020` and `CURR` with arguments Expression=@, Colours=never . . . 131
Table 5.13 `icgrep` PAPI normalized difference comparison between `2020` and `CURR` with arguments Expression=Email, Colours=never . . . 132
Table 5.14 `base64` PAPI normalized difference comparison between `FB64` and `CURR` 135
Table 5.15 `csv2json` PAPI normalized difference comparison between `SEQ` and `CURR` 136
Table 5.16 `icgrep -colours=always` PAPI normalized difference comparison between `UGREP` and `CURR` . . . 137
Table 5.17 `icgrep -colours=never` PAPI normalized difference comparison between `UGREP` and `CURR` . . . 138
Table 5.18 `u8u16` PAPI normalized difference comparison between `UTF8LUT` and `CURR` . . . 139
Table 5.19 `u32u8` PAPI normalized difference comparison between `UTF8LUT` and `CURR` . . . 139

List of Figures

Figure 1.1 Properties about various scheduling strategies 4

Figure 2.1 Simplified Parabix framework UML class diagram 11
Figure 2.2 Memory layout of a 3-stream streamset 16
Figure 2.3 Abstract linear pipeline with n stages (Adapted from [15]) 20
Figure 2.4 Pipeline parallelism models (Adapted from [79]) 21
Figure 2.5 Fixed-data pipeline execution with 3 threads (Adapted from [75].) . 21
Figure 2.6 Abstract pipeline stage definition 22
Figure 2.7 Kernel Lookahead input dependency 26
Figure 2.8 Simplified kernel builder UML class diagram 34
Figure 2.9 Vertical IDISA operations (Adapted from [55]) 35
Figure 2.10 Horizontal IDISA operations (Adapted from [55]) 36
Figure 2.11 Expansion IDISA operations (Adapted from [55]) 36
Figure 2.12 Field movement IDISA operations (Adapted from [55]) 36
Figure 2.13 Byte-space to Bit-space Transposition 41
Figure 2.14 Filter by mask example 42
Figure 2.15 Spread by mask example 44
Figure 2.16 Simplified Pablo compiler UML class diagram 45
Figure 2.17 Pablo program for `<[a-zA-Z]+>` 49
Figure 2.18 $R =$ `ScanThru`(M, C) (Adapted from [24]) 49
Figure 2.19 Pablo program for `[ab]*bbb[ab]*` using MatchStar 50
Figure 2.20 $R =$ `MatchStar`(M, C) (Reproduced from [27]) 50
Figure 2.21 Pablo program to find the first non-whitespace character 51
Figure 2.22 Pablo program to find tokens of $2 - 8$ non-whitespace characters . . 52
Figure 2.23 Non-validating Pablo program for Grammar 2.2 53
Figure 2.24 $R =$ `EveryNth`$(S, 2)$ 53
Figure 2.25 Pablo program for $\langle$index$\rangle\{2, 8\}$ using `IndexedAdvance` 54
Figure 2.26 $R =$ `IndexedAdvance`$(S, I, 2)$ 55
Figure 2.27 Pablo program for Grammar 2.3 (Adapted from [68]) 56
Figure 2.28 Pablo program for validation of Grammar 2.4 using `SpanUpTo` . . . 57
Figure 2.29 $R =$ `SpanUpTo`(S, E) 58

Figure 2.30 Pablo logarithmic length-sorted partitioning algorithm example with $n = 4$ . . . 59
Figure 2.31 Pablo PackL and PackH intrinsics . . . 62
Figure 2.32 Example of 3-to-2 Multiplexing / 2-to-3 Demultiplexing . . . 68

Figure 3.1 Simple example of SDF Graph . . . 71
Figure 3.2 Two SDF graphs with no admissible schedule . . . 72
Figure 3.3 SDF graphs with topology matrices . . . 73
Figure 3.4 Example of a streamset buffer subgraph . . . 75
Figure 3.5 Impact of looped schedules with stateful kernels . . . 76
Figure 3.6 Flowchart of the Evolutionary Algorithm (Adapted from [120]) . . . 78
Figure 3.7 Uniform order-based crossover (Adapted from [120]) . . . 79
Figure 3.8 Scramble sublist mutation (Adapted from [120]) . . . 80
Figure 3.9 Weighted interval graph (a) and corresponding interval colouring (b) (Adapted from [117]) . . . 81
Figure 3.10 Example of the transformation of a simplified Parabix dataflow graph to dependency and interval graph . . . 82
Figure 3.11 Example of G_0 when $m = 4$ (Adapted from [117]) . . . 84

Figure 4.1 Simplified example of partitioned `icgrep` program . . . 88
Figure 4.2 Example of a simplified Parabix dataflow dependency and interval graph . . . 95
Figure 4.3 Example DAWG for all traversals of Figure 4.3 . . . 95
Figure 4.4 Impact of schedules on partition jumping opportunities when a produces variable-rate data. . . . 98
Figure 4.5 Example of jump graph represented as sequence of nested intervals. 99

Figure 5.1 `base64` speed up compared to minimum single-thread cost . . . 112
Figure 5.2 Speedup of `base64` compared to ideal pipeline-parallel cost . . . 112
Figure 5.3 `csv2json` speed up compared to minimum single-thread cost . . . 113
Figure 5.4 Speedup of `csv2json` compared to ideal pipeline-parallel cost . . . 113
Figure 5.5 `editd` speed up compared to minimum single-thread cost . . . 114
Figure 5.6 Speedup of `editd` compared to ideal pipeline-parallel cost . . . 114
Figure 5.7 `gb18030` speed up compared to minimum single-thread cost . . . 115
Figure 5.8 Speedup of `gb18030` compared to ideal pipeline-parallel cost . . . 115
Figure 5.9 `icgrep -colors=never` speed up compared to minimum single-thread cost . . . 116
Figure 5.10 Speedup of `icgrep -colors=never` compared to ideal pipeline-parallel cost . . . 116

Figure 5.11 `icgrep -colors=always` speed up compared to minimum single-thread cost . . . 117
Figure 5.12 Speedup of `icgrep -colors=always` compared to ideal pipeline-parallel cost . . . 117
Figure 5.13 `u8u16` speed up compared to minimum single-thread cost . . . 118
Figure 5.14 Speedup of `u8u16` compared to ideal pipeline-parallel cost . . . 118
Figure 5.15 `u32u8` speed up compared to minimum single-thread cost . . . 119
Figure 5.16 Speedup of `u32u8` compared to ideal pipeline-parallel cost . . . 119
Figure 5.17 `ztf compression` speed up compared to minimum single-thread cost 120
Figure 5.18 Speedup of `ztf compression` compared to ideal pipeline-parallel cost 120
Figure 5.19 `ztf decompression` speed up compared to minimum single-thread cost . . . 121
Figure 5.20 Speedup of `ztf decompression` compared to ideal pipeline-parallel cost . . . 121
Figure 5.21 Average speed-up of selected platform compared to single-thread `SSE-4.2` execution for all applications except `csv2json` and `u32u8` using `CURR` . . . 122
Figure 5.22 Overall reduction in CPU cycles between `2017` and `CURR` . . . 124
Figure 5.23 Overall reduction in CPU cycles between `2020` and `CURR` with all applications except `icgrep -colours=never` . . . 129
Figure 5.24 Overall reduction in CPU cycles between `2020` and `CURR` for `icgrep -colours=never` . . . 130
Figure 5.25 Overall reduction in CPU cycles between `CURR` and `HYBRID` by program 134

Glossary

ACO ant colony optimization. 163, 164

API application programming interface. 19, 33, 37, 64, 123, 125, 126, 128

AST abstract syntax tree. 46, 47, 62

BDF boolean dataflow. 4, 85

BNF Backus–Naur form. 46, 53, 55, 57, 62

CC connected component. 152

CPU central processing unit. 19

CSV comma separated values. 1, 25, 106, 135

CUDA compute unified device architecture. 37, 39

DAG directed acyclic graph. 77, 78, 83, 96

DAWG directed acyclic word graph. 95, 100, 163, 164

DSA dynamic storage allocation. 80, 94, 100, 151–153, 155

DSP digital signal processors. 4, 5, 74, 75, 77

EA evolutionary algorithm. 94–97, 100, 102, 153, 163, 164

ETL extract transform load. iii, 1

FIFO first-in first-out. 2, 9, 20, 42, 70, 75

FPGA field-programmable gate array. 4

FSM finite state machine. 25, 48

GCD greatest common divisor. 74, 87, 91

GPGPU general-purpose computing on graphics processing units. 37, 43

HMCR harmony memory consideration rate. 152, 153, 155–160

IDISA inductive doubling instruction set architecture. xii, 1, 9, 35–37

IPC instructions per cycle. 101

IR internal representation. 1, 11, 12, 31, 33, 37–39, 45, 58, 60, 64, 65, 67, 123, 124, 126

JIT just-in-time. 2, 5, 10–12, 14, 16–20, 22, 29–34, 37–39, 64, 65, 86, 97, 104, 105, 123, 128

JSON javascript object notation. 1, 2, 53, 55, 106, 135

KPN kahn process network. 4, 5, 11, 73, 85, 91

LCM least common multiple. 74

LLC last level cache. 102

LLVM low-level virtual machine. iii, v, 1, 2, 8–11, 19, 30–33, 35, 37–39, 45, 47, 56, 64–67, 96, 103, 104, 123, 124, 126

OS operating system. 6, 7, 17, 24, 27, 30, 32, 39, 40

PAPI performance application programming interface. 104, 105, 123

PDF portable document format. 1

PTX parallel thread execution. 37

RE regular expression. 1, 2, 25, 28, 31, 32, 41, 50, 53, 62, 63, 107, 109, 110, 123, 130, 133, 136, 141

RPN reactive process network. 85

SCC strongly connected component. 77

SDF synchronous dataflow. 3–5, 70–78, 86–89, 94, 135, 168

SIMD single instruction, multiple data. iii, 1, 2, 9, 24, 35–37, 43, 44, 48, 65, 108, 122, 168

SISO single-input single-output. 14, 20, 85

SWAR SIMD within a register. 35

UML unified modeling language. xii, 11, 34, 45

UPO uniquely partially orderable. 83, 84

VRDF variable-rate dataflow. 5, 89

XML extensible markup language. 1, 57, 63

Chapter 1

Introduction

As businesses became increasingly dependent on the data analytics, Big Data turned into one of the primary drivers of computer performance. Recently, Big Data began shifting away from pure ETL solutions and towards real-time analysis of "raw text" [1, 57, 62–64, 83, 91]. Here, raw text can refer to data of any format, including fully structured SQL databases, semi-structured CSV, JSON or XML documents, unstructured free form text such as emails or PDFs, and even binary data like images, audio and compressed archives.

In this dissertation, we introduce the Parabix dynamic-compilation framework. Parabix is designed to generate high-performance user-query-driven text-parsing programs, which can be used as enriched scan operators for Big Data systems. Our goal is provide a framework that maximizes the multi-thread acceleration of Parabix programs using simple pipeline parallelism. Although this framework targets the needs of a typical "Parabix-style" SIMD-dominated application (such as those found in [24, 25, 27, 76]) it supports any program that can be expressed with LLVM IR that adheres to our I/O processing model. With the remainder of this chapter, we discuss exactly what this means, and introduce the core concepts behind the Parabix framework and dataflow network programming before concretely laying out the research objective and limitations of this dissertation.

1.1 Brief overview of the Parabix framework

Parabix is an intrinsically bit-parallel programming paradigm, consisting of three interconnected components: the IDISA library [26, 55], the `Pablo` programming language [24, 27] and the Parabix framework [75]. IDISA is a generic SIMD library that facilitates the direct construction of vectorized LLVM code. Programs that leverage IDISA are primarily written in `Pablo`, which is a highly-restricted domain-specific language. Further discussion of IDISA and `Pablo` can be found in §2.1.8 and §2.3, respectively. In spite of `Pablo`'s restrictive nature, with it we have developed a Unicode `UTF-8` to `UTF-16` transcoder [23], a XML validator [25], a basic regular expression (RE) matcher [27], an edit distance matcher [75] and accelerated the Xerces-C 3.1.1 XML parser [81]. Arguably, the techniques integral to

`Pablo` can be found in the Mison JSON [74] and `simdjson` [68] parsers and Parabix is well suited for a variety of data filtering techniques found in literature, such as Sparser [93].

The Parabix framework is a `C++` library that simplifies the creation and dynamic compilation of highly-parallel programs [34, 75] and is intrinsically tied to the LLVM JIT-compilation engine, `MCJIT`. Dynamic compilation is a requirement of any Parabix program that generates programs based on user input, such as our RE matcher, `icgrep`. Although the run-time cost of Parabix programs tends to be dominated by executing code in `Pablo` kernels, not all portions of any Parabix program can be written strictly with `Pablo`. For example, `Pablo` cannot perform file I/O nor dynamically allocate memory; these (amongst others) are intended omissions to prohibit non-SIMD work by `Pablo` functions. To support a wider variety of programs, the Parabix framework contains a wide variety of "non-SIMD" kernels and the ability to develop new custom kernels. Herein we define **kernel** to be a side-effect-free function that communicates with other kernels strictly through shared memory buffers that we refer to as **streamsets**. Because we want to maximize the amount of compile-time knowledge available to the framework's internal compilers, our framework supports a variety of I/O processing rates and attributes. These allow us concretize the data access patterns between kernels and streamsets. Of key importance to this section are the `Fixed`, `Bounded`, `PopCount` and `Greedy` processing rates and the `LookAhead` attribute. `Fixed` rates produces n tokens per invocation and `Bounded` rates produce n to m. `PopCount` are a specialization of `Bounded` that allows the subprogram to know a priori how many tokens will be transferred prior to invocation and `Greedy` rates consume any enqueued input token upon kernel invocation but may optionally have a minimum lower bound. Finally, a `LookAhead`(k) attribute indicates that a kernel must peek k tokens ahead of its current token to correctly execute. We discuss these and others in detail in §2.1.2 and §2.1.6, respectively.

Currently, the Parabix framework generates **linear-pipeline programs** [75]. Pipelining is well-known technique wherein a program is decomposed into discrete stages (kernels) that communicate with each other strictly via buffers (streamsets). With linear pipelines, a static sequence of stages is repeatedly looped over until some termination condition is met. Linear-pipeline programs are trivially parallelizable but place hard restrictions on the programmer to ensure correctness [80]. We continue this discussion and describe our implementation in §2.1.5.

1.2 Dataflow models

Beginning with Karp and Miller's seminal work on computation graphs [61], dataflow programming is a paradigm designed to simplify the development of parallel systems. With it, programs are modelled as directed multigraphs; each dataflow graph consists of a collection of nodes connected via channels. Each *node* represents a deterministic side-effect-free function and each *channel* is a unidirectional FIFO queue in that transfers a sequence of data

from a producing node to a consuming node. Any communication between nodes is strictly through one of its channels. An I/O *port* connects a node to a channel and each channel is between precisely two ports. Each datum, referred to as a *token*, is restricted to a single channel-specific type (e.g., integers, strings, objects, functions, etc). Associated with each port is a *rate* that indicates how many tokens are transferred through the port (i.e., pushed to or popped from the channel) at each invocation of a node. During an initialization phase, channels may be enqueued with initial tokens; these tokens are commonly referred to as *delays*. Every cycle in a valid dataflow graph has at least one delayed channel [61].

With few exceptions (e.g., RPNs [44]), dataflow graphs describe deterministic programs. Determinism is assured because the sequence of token values flowing through every channel is provably independent of the node execution order [41, 59]. However, preserving this property means a node cannot choose its behaviour based on the availability of its input(s) [59, 61]. For example, a *merge* node with two input channels and one output channel that enqueues its first available input token on its output channel is considered non-deterministic due to an implicit dependence on time [18].[1] Despite this limitation, many dataflow models themselves are known to be Turing-complete [20].

Coarsely speaking, dataflow programs can be divided into two major classes: *static* and *dynamic*. Static dataflow models require that all I/O rates between nodes are fixed and known at compile time whereas dynamic models permit rates to change at run-time. The primary advantage of static dataflow models is that they define the space of dataflow programs where a static schedule (defined shortly) may be generated, maximum channel length is bounded, and termination (or non-termination) can be proven. Dynamic models are considerably more expressive and can cover a far wider range of program but require some form of run-time arbitration, such as dynamic memory management, to execute correctly.

A wide spectrum of dataflow models exist, ranging from fully static to fully dynamic. Each new class attempts to better demarcate the region between the two extremes. Figure 1.1 lists the pertinent commonly-accepted categories. Fully-static models, such as synchronous dataflow (SDF) [71], represents the class of program in which every scheduling decision about them can be made at compile-time. Scheduling, in this context, refers to a fixed sequence of node executions, which we refer to as *invocations*. Although the problem is NP-Complete [86], in theory, it is possible to compute an optimal schedule for a fully-static program w.r.t. throughput and/or memory requirements. Fully-static programs require no run-time interventions (e.g., dynamic memory management or thread synchronization) beyond executing the computed schedule. However, fully static systems require that every

[1]The only permissible notion of a merge node within any dataflow model is one in which every token has a unique timestamp that controls the order in which all nodes process data, every node shares a synchronized notion of time, and the execution time of a node is purely dependent upon its channel histories and the tokens currently being processed by each node at the start of its current invocation [18].

aspect of the program — including precise information about invocation latency — is defined a priori. Consequently, it is typically reserved for DSP or FPGA circuits where each node represents a very small code fragment — often a single operation. Since this tends to be too restrictive, self-timed scheduling is usually considered the lower bound for any complex system. Self-timed scheduling only requires that the I/O rates of each node are fixed and known at compile time. Like fully-static models, a fixed schedule and memory allocation strategy can be computed but some form of synchronization is required to mitigate any communication delays during cross-thread token generation / transfers.

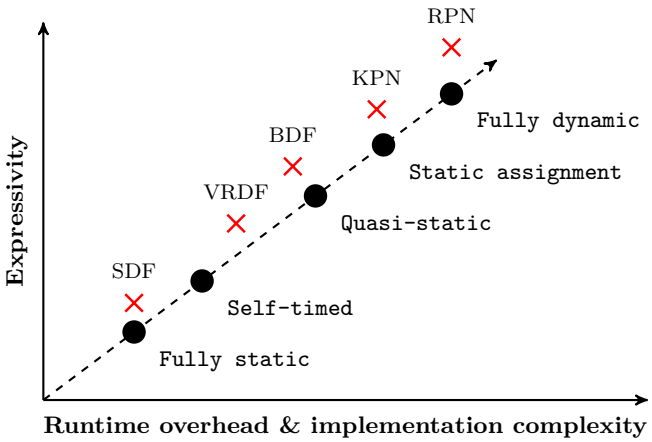


Figure 1.1: Properties about various scheduling strategies

Quasi-static scheduling is the least expressive dynamic scheduling model but the boundary between it and static assignment is often unclear. Both models permit input-dependent node behaviour but whereas quasi-static scheduling tends to impose some constraints to guarantee that something proximating an static schedule can be generated, static assignment simply maps nodes to processors and leaves the responsibility of scheduling nodes to a run-time manager. To differentiate between invocations in a schedule and self-determined node executions in a static-assignment system, we refer to the latter as *firings*. Conceptually, any node can fire (or be invoked) once it is *enabled*, which occurs when it receives sufficient input. A classic example of quasi-static scheduling is the boolean dataflow (BDF) model [20], which incorporates control flow mechanisms into the SDF model [42]. On the other hand, pure implementations of Kahn process networks (KPNs), such as RaftLib [8], are good examples of static assignment. Finally, fully-dynamic models, such as RPNs [44], defer all scheduling decisions to run-time but may include models that can react to user events and even permit mutable network topology.

1.3 Contrasting traditional and Parabix dataflow models

Although the dataflow model and the kernel and streamset model employed within Parabix are similar, there are many significant differences between the two that makes combining them difficult. In traditional dataflow models, channels typically abstract register-to-register datum transfers between two stateless microcode fragments. In the Parabix model, kernels are complex stateful functions, typically ranging from 10s to 100,000s of instructions in length. This complicates leveraging existing dataflow literature in two ways: firstly, to better leverage the I-cache, kernels tend to process large streams of data every invocation. Consequently, I/O streamsets are often very large, typically $4-64$ KB $\times$ number of threads each. More importantly, to avoid unnecessary `memcpy` operations, streamsets may have many consumers, lending to the notion of a *shared buffer*. The second complication is related to the first; because memory is prohibitively expensive in DSP systems — often targeted by dataflow systems — emphasis is often placed on obtaining schedules that minimize memory consumption, either through memory-reuse/register-allocation strategies [87–89, 101] or factoring [10–14, 53, 90, 99]. Memory in commodity computers is abundant, meaning such concerns ought to be ignorable. Consequently, *the primary objective of traditional dataflow scheduling algorithms is incompatible with the needs of the Parabix framework.*

To the best of our knowledge, no existing dataflow model, scheduling algorithm or framework considers shared buffers exactly as we require them here. The only work to directly address this subject in literature is by Denolf et al. [35], who present a method of proving that a "shared channel's" memory is sufficient to permit a periodic execution of a program but do not discuss its implementation. Similarly, even though several frameworks permit shared buffers as we desire them, they either do not factor in their use prior to buffer allocation [16, 32, 49] or do not have an identifiable scheduling policy [106].

The second difficulty stems from the fact that Parabix targets text parsing problems. This presents a major hurdle when adopting static dataflow techniques because even for regular languages, tokenization alone cannot transform the problem into one that permits static models. Quasi-static dataflow models allow non-fixed I/O patterns [20, 113] but are too restrictive to efficiently support the needs of a general multi-threaded program. For such software, dynamic models, such as KPNs [59] could be used but implementations of KPNs tend to focus on token-by-token data transfers and often mimic fixed-code linear-pipeline programs [8, 37, 92, 106], which Dan Lin showed to be suboptimal for Parabix programs [75].

With the major differences acknowledged, we can now discuss the similarities between the Parabix and static dataflow models. `Fixed` and `PopCount` processing rates have a one-to-one correspondence with the constant and symbolic rates used by the SDF static [71] and variable-rate dataflow (VRDF) quasi-static models [113]. Similarly, a `LookAhead` of k implies a delay of k on a channel. Unfortunately, `Bounded` rates, by their definition, cannot be predicted at JIT-time and thus entail some form of dynamic model. Finally, `Greedy` rates

is an interesting case in which it may be static or dynamic depending on the value of its lower bound: if and only if the output rate of a streamset producer is guaranteed to support it, can the rate be reasoned about statically.

Since the majority of Parabix programs contains `Bounded` or `PopCount` processing rates, finding a way to maximize the utility of static scheduling within a dynamic network is preferable. Lee and Messerschmitt conjectured that a good way to handle this problem is to partition a dataflow graph into a set of consistent subgraphs connected by non-synchronous channels and schedule each subgraph according to some run-time protocol [71] but they did not describe how to identify the set of maximal consistent subgraphs. The current version of their framework, Ptolemy II, supports mixed-domains but places the burden on the programmer to organize a program into discrete hierarchical components [38]. While Gu et al. automated this, their algorithm assumes that fixed and variable rate I/O cannot coexist on the same layer [50] — a common case in Parabix programs. Consequently, although there are many surface-level similarities between the Parabix and dataflow models, correctly combining such models has not been addressed in literature.

1.4 Research objective

Our goal is to develop a framework that reliably provides a **near-linear multi-threaded speedup for linear-pipeline programs on commodity hardware**. Pipeline parallelism prevents us from efficiently leveraging more than $\lfloor {}^{total\ single-threaded\ time}/_{\max(total\ kernel\ time)} \rfloor$ threads but many issues will prevent programs from fully utilizing them. Some problems are inherit to the program, OS and/or user input data but others can be mitigated. By organizing data with hardware prefetching in mind, we can help minimize false sharing and cross-core latency, both of which degrade the performance of multi-threaded applications.

Supporting this objective, we make the following contributions to the framework:

1. We expand on the current kernel and streamset model by incorporating additional processing rates and kernel and I/O port attribute annotations. Our hope is to ease burden of creating efficient multi-threaded programs by abstracting the implementation details from the programmer but providing the framework with enough information to make correct decisions regarding program schedule and memory layouts.

2. We transform the per-program-execution dynamically-compiled pipeline into a specialized kernel subclass, permitting it to use the same caching infrastructure as the other kernels and providing the ability for nested pipeline constructs.

3. We leverage dataflow programming by clustering groups of kernels into tightly interconnected partitions whose I/O rates can be statically computed, reducing the run-time overhead.

Apart from the central question, this dissertation also addresses the following secondary questions:

`Q1`: What aspects of dataflow programming can be used to analyze and reduce a program's memory footprint?

`Q2`: How does improvements in hardware affect single and multi threaded performance?

`Q3`: Can static dataflow concepts meaningfully reduce the pipeline computation overhead?

`Q4`: Can hybrid linear-pipeline parallelism further accelerate our programs?

1.5 Limitations

This dissertation focuses on the acceleration of a single static linear-pipeline application on 64-bit Intel architectures running POSIX-oriented OSs with general-purpose techniques. Coscheduling applications is an incredibly complex topic that falls outside of the scope of this dissertation; we refer the interested reader to Schönherr's doctoral dissertation [104].

Dynamic scheduling approaches that incorporate work-stealing have been shown to improve performance in load-imbalanced programs [78]. Parabix utilizes a fixed-data pipeline parallelism model (§2.1.5) which is considered to be inherently load balanced [15] but as the results for `icgrep -colours=always` shows in Ch. 5, there is room for improvement. We briefly discuss alternate methods of thread-based parallelization in §2.1.5 but they too fall outside the scope of this dissertation.

We focus on exploiting hardware memory prefetchers but acknowledge that software prefetchers may provide additional benefits. Early experimentation into the use of software prefetchers found it difficult to place the prefetch instructions at a position that provided noticeable performance improvements but this was not an exhaustive study. Similarly, because kernels process long sequences of data, fine-grained cache-usage techniques such as cache-conscious data placement [22] and slice-aware memory management [40] were not fully investigated. Such strategies focus on the placement of small objects (i.e., those less than a cache-line size) which does not match our expected memory model. Instead we focused on cache-friendly memory layouts, discussed in §4.4, but acknowledge that some kernel state objects could benefit from more advanced memory placement policies.

1.6 Contributions

The major contribution of this dissertation is the pipeline compiler and the additions to the supporting framework. Originating with the work of Robert Cameron on `UTF-8` to `UTF-16` transcoding in 2007 [23], the Parabix framework was aimed towards creating static console applications. Following the work of Dale Denis in 2014, Parabix transitioned from

a statically-compiled programming model to one that exploited the dynamic-compilation capabilities of LLVM [34]. In 2017, Dan Lin further improved the framework, incorporating a shared buffer dataflow model and linear-pipeline parallelism to enabled programmers to easily generate multi-threaded Parabix programs without diminishing any of the data-parallelism inherit in the existing system [75]. Although this version satisfied the functional requirements of our then-current programs, our goal of supporting a wider variety of programs necessitated the addition of a more general notion of processing rates and attributes, which we discuss in §2.1.2 and §2.1.6. Furthermore, the current pipeline compiler absorbs all of the burdens for buffer management, kernel sequencing, and dataflow shaping that were originally placed upon the programmer. Other components of the current framework, such the `Pablo` optimization passes (§2.3.4) were developed with the goal of improving general program performance and reducing programmer work.

Apart from the identifying an unexplored area of dataflow research, to the outside research community, the primary contribution of this dissertation is the partitioning and the scheduling algorithms that we present in Ch. 4. These algorithms leverages the strengths of the traditional dataflow models whilst satisfying the needs of the Parabix programs: specifically, support for variable-rate I/O and shared buffers in a low-overhead linear-pipeline program. Finally, we introduce the concept of partition jumping in which the analysis phase of the pipeline compiler can reason out which kernels can be safely skipped when a predecessor is found to have insufficient input for processing and generates schedules that maximize the utility of this observation.

1.7 Thesis structure

The remainder of this dissertation is structured as follows:

- Ch. 2 discusses the Parabix framework, describing many of the high and low level concepts necessary to develop Parabix programs. It addresses both contribution 1 and 2 of §1.4. Although this chapter is not intended for the wider general audience, discussions within later chapters assume the reader is familiar with §2.1.1 – §2.1.3 and §2.1.5.

- Ch. 3 reviews dataflow programming, focusing on the aspects of it that were found to be useful in the development of this framework.

- In Ch. 4, we discuss the pipeline compiler used by the framework, focusing on dataflow schedule generation; this addresses contribution 3 of §1.4 and secondary question `Q1`.

- Ch. 5 presents our evaluation of the programs developed for this framework, comparing it to previous versions of the system as well as several state of the art programs to determine how close we come to achieving research objective and answering `Q2` – `Q4`.

- Finally, Ch. 6 concludes the dissertation and details recommendations for future work.

Chapter 2

Description of the Parabix framework

In this chapter, we present the current state of the Parabix ecosystem. Note, this chapter assumes — but does not require — some familiarity with LLVM and basic programming and compiler terminology. Any required knowledge will be explicitly defined. Further information on other interesting topics will provided via hyperlink or citation to existing literature.

Parabix is an intrinsically bit-parallel programming paradigm, consisting of three interconnected components: ① the IDISA library [26, 55], ② the `Pablo` programming language [24, 27] and ③ the Parabix framework [75]. IDISA is a generic SIMD library that abstracts many of the functions required by the framework. `Pablo` is a highly-restricted domain-specific language, tailored for text parsing problems. However, not all portions of a Parabix program can be written in `Pablo` (e.g., `Pablo` cannot perform file I/O nor allocate memory.) These (amongst others) are intended omissions to prohibit non-SIMD work by `Pablo` functions. To support a wider variety of programs, the framework contains a wide variety of kernels to facilitate some of the non-SIMD as well as the ability to develop new custom kernels. The term **kernel** comes from Parabix's dataflow roots; we define kernel in §2.1.1 but for now assume they are side-effect-free functions. Kernels can be stateful or statefree and communicate with each other through **streamsets**. We define streamsets in §2.1.3 but they can be viewed as FIFO buffers used to transmit data between kernels.

Finally, three important terms will appear throughout in this chapter: segment, stride and block_width. Each time a kernel is executed, it conceptually processes one **segment** of data. A segment is composed of a potentially-variable number of strides. Each **stride** is essentially an atomic unit of work performed by a kernel at run-time. The *stride_length* is a fixed property of each kernel's definition and typically equal to some multiple of the **block_width**, which is equal to the SIMD-register (i.e., vector) bit width of the current architecture (e.g., 128/256/512-bit for SSE, AVX and AVX-512 systems, respectively.)

2.1 High-level design

The Parabix framework is a `C++` library that facilitates the construction of highly-parallel programs [34, 75]. Because this framework is intrinsically tied to the LLVM JIT-compilation engine, we briefly introduce the concept of "JIT-ing" before discussing the framework itself.

Just-in-time (JIT) dynamic compilation has a long history in computer science [6]. Essentially, programs that utilize JIT-compilation defer some of their compilation work to program execution time aka the run-time phase of their life cycle. This provides such programs the ability to optimize their core code fragments for the specific hardware they are operating on. To be explicit, JIT-ing is not synonymous with interpreting code; JIT-ed code is compiled to architecture-specific machine code and then executed at run-time. Although the particulars of JIT compilation goes beyond the scope of this dissertation, it is useful to consider JIT-ing in general when discussing a Parabix program because the meaning of different aspects of the program changes depending on its current life cycle stage.

The three major program life cycle phases to consider are: edit-time, compile-time and run-time. *Edit-time* is when a programmer writes the static source code; *compile-time* occurs as that code is compiled into an executable program and *run-time* transpires whenever a user executes that program on their system. We further subdivide the run-time phase into start-up, JIT-compilation, and JIT-execution subphases, indicating the portions of the run-time devoted to ① invoking the statically-compiled code for the purpose of generating dynamic code; ② JIT-compiling any dynamically-generated code and ③ running the "JIT-ed" code. Informally, we herein *refer to statically-compiled program code as `C++` code and dynamically-compiled code as JIT-ed code* when the distinction is not readily apparent.

During run-time, Parabix programs may interleave multiple subphase sequences in various ways. For example, when recursively searching files, `icgrep` compiles a new filter kernel whenever it encounters a `.gitignore` file. Similarly, `editd` utilizes a preparsing phase to improve the performance. In both examples, the programs moved between subphases repeatedly. For simplicity, we assume programs execute each subphase once.

Figure 2.1 depicts a slightly simplified view of the critical components of the current architecture. In the following subsections, we will discuss each component in detail. We do not provide any code in these sections; instead we focus on the intentions and design decisions behind each class. Code artifacts are available at:

`cs-git-research.cs.surrey.sfu.ca/cameron/parabix-devel`.

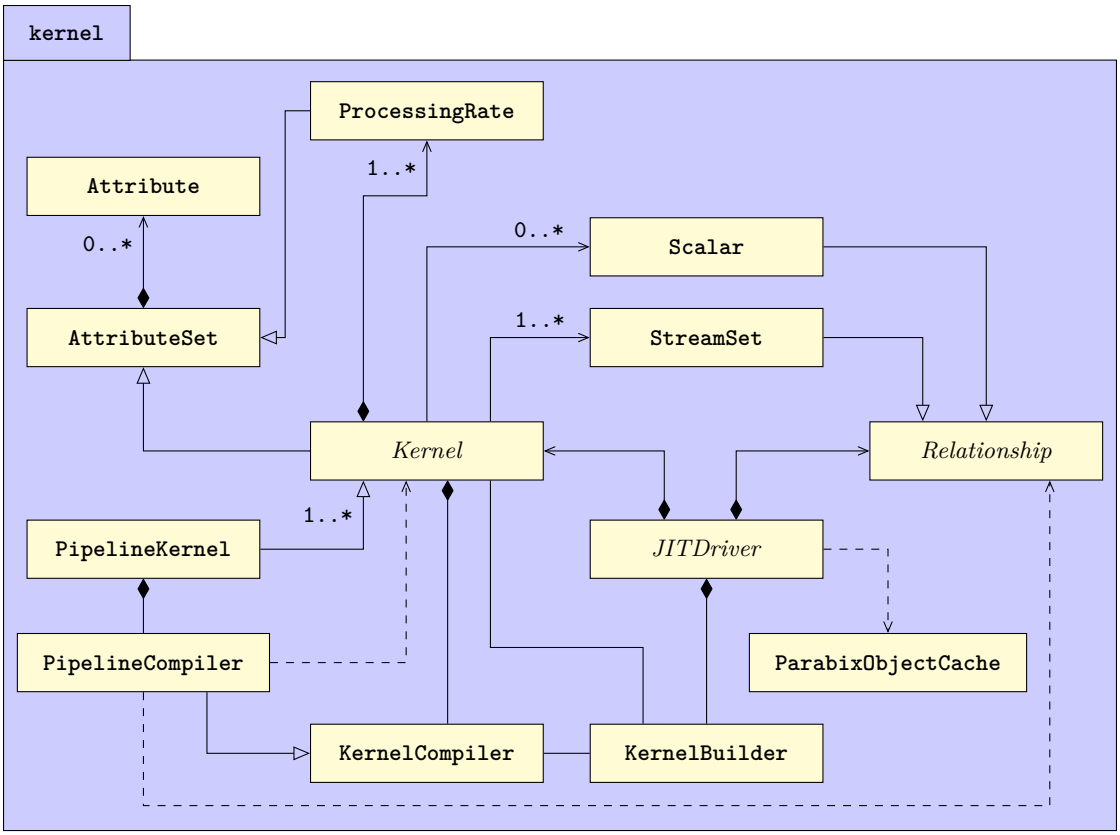


Figure 2.1: Simplified Parabix framework UML class diagram

2.1.1 Kernels

At edit-time, a programmer first defines the kernel classes, which are then instantiated at start-up. Each kernel is an independent object that cannot access any unowned memory at JIT-execution time except through its **I/O ports**. We use the term *port* here abstractly to mean any placement of code involving data interchange between a kernel and streamset[1]. Most kernels are *stateful*, meaning that the value of any internal scalar values (described in §2.1.4) persist between per-segment invocations. Kernel state cannot be modified by any other kernel except indirectly via the transference of data through its input ports.[2]

Once the programmer instantiates the kernels, they can define the kernel I/O relationships by linking them to common **streamset** buffer objects (defined in §2.1.3.) These

[1]Although the framework does not currently support distributed deployment across multiple systems, this is not a fundamental limitation. Proof of this statement can be seen in the theory behind KPNs [59,60].

[2]Obviously, given kernels generate LLVM IR, these restrictions could be worked around using global memory or other means but the framework will not make any accommodations for such violations. It is the programmer responsibility to ensure the program correctly executes in the context of our parallelization model (§2.1.5).

restrictions help ensure a deterministic flow of data between kernels [59] and serves as the foundation of our thread-safe parallel processing model, discussed in §2.1.5.

At edit-time, it is crucial to view kernels as **generators** (or compilers) for code that will be created during JIT-compilation. Common to many dataflow frameworks [2, 17, 48, 105, 106], each kernel is a `C++` class that inherits from a common `kernel::Kernel` base class, which (abstractly) contains three overridable methods: `DoInit`, `DoSegment` and `DoFinalize`. Each method corresponds to a JIT-execution subphase and of the three, `DoSegment` is often the most complex of the methods since its logic encapsulates the "work" function generator of the kernel. Each time a kernel performs "work" during JIT-execution, it conceptually processes one **segment** of data, lending to the name of the `DoSegment` method. As discussed shortly in §2.1.2, associated with every I/O port is a `ProcessingRate`, indicating how much data/space is required by a kernel to process one segment. The pipeline (discussed in §2.1.5) will only call the kernel's `DoSegment` method during JIT-execution when every `ProcessingRate` has been satisfied — with one critical exception: ignoring the complication of external I/O, every streamset is written to by exactly one kernel, i.e., every streamset has exactly one producer. Streamsets are considered to be unbounded sequences of data until its producer terminates, at which point the streamset is deemed closed. Barring the appropriate input rate attributes, discussed in §2.1.6, once a kernel reads from a closed input with insufficient data for a single stride, it is placed into a **final-mode** state in which it will be provided with any available data and permitted to execute a segment consisting of exactly one partial stride.

Although only `DoSegment` must be implemented by the programmer, the IR any of its overridable methods generate can differ based on the parameters passed into the `C++` kernel class at instantiation, architecture characteristics or — despite the JIT-execution-time preclusion — settings stored in global memory. Each method corresponds to a phase of the pipeline life cycle. We continue discussing these methods in §2.1.5 after introducing a few important concepts.

2.1.2 Processing rates

To concretize the data access pattern, each kernel **I/O port** is assigned a `ProcessingRate`. This rate defines how many items are either processed or produced by the kernel each stride. For simplicity, we refer to processed/produced items as being *transferred* through the I/O port. The rates currently supported by the framework are:

- **`Fixed:`** an I/O port with a fixed rate of n rate will transfer $n \times$ *stride_length* items per stride. Fixed-rate ports are the simplest from an analysis point of view but require special consideration during the final-mode of JIT-execution, which we discuss at the end of this subsection. Note also that although fixed rates permit rational numbers, the total number of items per stride must be a natural number.

• **`Bounded:`** a bounded $n-m$ rate states the kernel transfers between $[n, m] \times \textit{stride\_length}$ items per stride. A kernel must explicitly set its total processed/produced item count during the segment to inform the pipeline how many items were successfully transferred.

• **`PopCount:`** this rate is a specialization of the `Bounded`$(0, 1)$ rate and a unique feature of the Parabix framework. Each `PopCount` input port has both a source and reference streamset. The source is the streamset in which data is transferred from at a rate indicated by the value of the reference streamset. Specifically, at each $i^{\texttt{th}}$ stride of a kernel $\mathcal{K}$ with a `PopCount` port, the number of items transferred is exactly the sum of the number 1-bits up to and including the $i^{\texttt{th}}$ stride-unit of the the reference streamset. A stride-unit is dependent on $\mathcal{K}$'s *stride_length* and I/O rate of the `PopCount` port. `PopCount` output ports produce data at the rate indicated by their reference stream but are not linked to a source streamset; instead $\mathcal{K}$ is expected to produce the exact quantity per stride-unit.

Although replacing `PopCount` rates with `Bounded` rates is always permissible, the pipeline must make a conservative choice as to how many tokens a kernel requires and would limit the number of strides a kernel executes to whatever satisfies its upper bound. This can affect thread-balancing by erroneously introducing stalls even though the required data is available. `PopCounts`, however, introduce a hidden cost to each program: internally, the `PipelineKernel` $\mathcal{P}$ inserts a `PartialSumKernel` $\mathcal{S}$ into the program for each unique reference streamset of a `PopCount` port. The output of $\mathcal{S}$ is read by $\mathcal{P}$ to determine the number of items transferred at each *segment_length* boundary. Because multiple kernels could share $\mathcal{S}$, whenever the *segment_length* of one is divisible by another, $\mathcal{S}$ will be generated for the smallest *segment_length* and $\mathcal{P}$ will extract the appropriate value for $\mathcal{K}$. An important note here is because $\mathcal{S}$ is also susceptible to *final-mode* processing, when the maximum step factor of its consumers exceeds one, $\mathcal{P}$ will automatically splat the final partial-sum value across the region of the streamset that will be accessed by them.

To reduce the number of `popcnt` instructions and horizontal (inter-lane) summations, a sequence of half-adders is used to reduce the total number to $\log_2\left(\frac{\textit{segment\_length}}{\textit{vector\_length}}\right)$. Although this helps to minimize the cost of employing `PopCount` rates, using them still incurs a small but non-negligible cost.

• **`Relative:`** indicates a port transfers data at constant ratio of some other port. An input port can only be relative to another input port but an output can be relative to either an input or output. To minimize the start-up resolution cost, it is considered an error for the reference stream of a `Relative` rate to be a `Fixed` or `Relative` rate port.

• **`Greedy:`** a `Greedy` rate may only be applied to an input port. Like its name implies, every `Greedy` port is expected to consume all data provided to it but can optionally have a lower bound indicating the minimum number of unprocessed items required by the kernel before it can be invoked. A kernel must have at least one non-`Greedy` input, a `Greedy` input

with a non-zero lower bound or must explicitly terminate itself upon completion (with the appropriate attribute set, discussed in §2.1.6) to be accepted by the pipeline.

- **`Unknown`:** an `Unknown` can only be applied to an output port. `Unknown` rates indicate that the number of items produced by a kernel has no known upper bound but may optionally have a known lower bound. The producing kernel is responsible for all memory allocations associated with that linked streamset.

	`Fixed`	`Bounded`	`PopCount`	`Relative`	`Greedy`	`Unknown`
`Countable`	✓	✗	✓	?	✓	✗
`Calculable`	✓	✓	✓	?	?	✗

Collectively, `Fixed`, `PopCount` and `Greedy` rates are known as countable rates whereas `Fixed`, `Bounded` and `PopCount` are considered calculable. **Countable** rates are those in which the pipeline (defined in §2.1.5) can determine precisely how many items will be transferred per segment *prior to* invoking a kernel's `DoSegment` method at run-time. **Calculable** rates are those in which the pipeline can *statically* determine the upper bound on the transfer-rate. Relative rates inherit the properties of their reference rate but the calculability of greedy rates is dependent on the output rate of its streamset.

> **Note:** except in the case of an early termination, the pipeline assigns the number of items transferred through each `Countable` I/O port after kernel invocation. For any non-`Countable` / non-`Relative` port, the programmer must explicitly call the `KernelBuilder`'s `setProcessedItemCount` or `setProducedItemCount` functions to inform the pipeline as to the I/O port's current state.

During JIT-compilation of the pipeline, processing rates help to define and constrain the problem of efficiently transferring data between kernels during JIT-execution. However, it is considered undefined behaviour to assign a rate to a port that conflicts with its data access pattern within its associated `DoSegment` method. Any I/O contract violation all but assures some form of data corruption and/or a segmentation fault at run-time.

Complication with final-mode processing for fixed-rate dataflow

Conceptually, fixed-rate dataflow is the simplest form of I/O within the Parabix framework but this simplicity hides a major complexity: *input data is rarely stride aligned.* After exhausting its input, a kernel will enter the final-mode state. During final-mode processing, a kernel will execute exactly one partial stride. When a kernel has both fixed-rate inputs and outputs, the number of items produced by these outputs is always a fixed ratio of of its shortest input relative to its input rate. For example, given a SISO kernel with a `Fixed`(a) input that has processed x items and `Fixed`(b) output rate, the total length of the output streamset is $\lceil x \cdot b / a \rceil$ — but what about kernels with multiple inputs of differing fixed-rates?

Let $a_1, a_2, \ldots, a_n$ be the rates of the fixed-rate inputs and $x_1, x_2, \ldots, x_n$ be the number of input items transferred through the port at the point of final-mode processing. Similarly let $b_1, b_2, \ldots, b_m$ and $y_1, y_2, \ldots, y_m$ be the rates and total length of the fixed-rate outputs, respectively. If we assume these variables were unbounded real numbers, we can compute the y-values with Eq. 2.1 but because we are computing these with 32/64-bit integers, we instead use Eq. 2.2 to correctly handle the cases when $(a_i \mod x_i) \neq 0$. We mitigate the possibility of integer overflow errors in Eq. 2.2 due to the lcm factor by limiting the x-values to refer to only *unprocessed* number of items currently in the streamset buffer and allow the y-values to refer to the number of *writable* positions for the final stride.

$$y_j = \left\lceil b_j \cdot \min_{i=1}^{n} \left(\frac{x_i}{a_i} \right) \right\rceil \tag{2.1}$$

$$y_j = \left\lfloor \frac{b_j \cdot \min_{i=1}^{n} \left(\frac{x_i \cdot \ell}{a_i} \right) + \ell - 1}{\ell} \right\rfloor, \text{where } \ell = \text{lcm}(a_1, a_2, \ldots, a_n) \tag{2.2}$$

Attributes — listed in §2.1.6 — can be used to modify processing rates and many can affect Eq. 2.2. The `Principal` attribute limits the set of a and x-values to consist only of the principal input streamset. `ZeroExtended` inputs are special in that their a and x values are considered only if their producing kernel has not terminated prior to the final-mode handling of the current kernel. `Add` and `Truncate` are perhaps the most subtle complications. Such attributes are common in `Pablo` kernels, discussed in §2.3, since many parsing problems requires a marker to be placed after the `EOF` to indicate a match (or error) on the final line. Not only can the raw x-values be directly modified by these attributes when they are associated with the kernel's input port but can also affect the buffer space requirements of consuming kernels when produced at such rates.

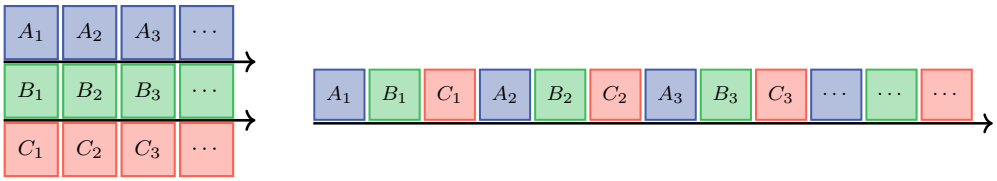
When a kernel enters final-mode processing, it must produce all of its output data prior to termination. If a streamset is both full and mapped to a static buffer, it must have an overflow region sufficient to satisfy its space needs. Overflow calculation is a transitive program property, requiring global awareness of `Add`, `Truncate` and `Principal` attributes. Given a topological traversal of the kernels in the program graph, Alg. 1 computes the table $\mathcal{T}$, which details how many items could be added to a streamset's overflow.

Algorithm 1 Transitive add calculation

$I \leftarrow$ the set of all input ports of kernel K
$P \leftarrow$ the `Principal` input port of K or I if none exists
$MinK \leftarrow$ **if** $P \neq \emptyset$ **then** ∞ **else** 0
for each $p \in P$ **do**
    $k \leftarrow \mathcal{T}_{\text{StreamSet}(p)} + \text{Add}(p) - \text{Truncate}(p)$
    $MinK \leftarrow \min(MinK, k)$
$O \leftarrow$ the set of all output ports of K
for each $p \in O$ **do**
    $k \leftarrow MinK + \text{Add}(p) - \text{Truncate}(p)$
    $\mathcal{T}_{\text{StreamSet}(p)} \leftarrow k$

2.1.3 Streamsets

At edit-time, I/O relationships are defined by binding each kernel I/O port to a streamset object. Each streamset represents a memory buffer that will be allocated during JIT-execution. At run-time, each kernel directly reads from/writes to a streamset buffer. Every streamset is written to by precisely one kernel but may be read by many consumers, leading to the notion of **shared buffers**. In accordance with both the `Pablo` and dataflow models, streamsets are viewed as arrays of m unbounded data streams. Each stream is an ordered sequence of w-bit width items. Kernels transfer items in *block_width*-length chunks, referred to as **elements**, and typically read one element of each stream per stride. For example, in Figure 2.2a we have a streamset composed of three streams A, B and C; we expect element A_1, B_1 and C_1 to be read before A_2. Whenever $m > 1$ sequential prefetchers tend to perform poorly [56]. Consequently, memory is for each streamset is laid out according to its expected access pattern, shown in Figure 2.2b. This technique is known as strip-mining and is a recommended locality optimization by Intel [58].



(a) Conceptual layout (b) Actual layout

Figure 2.2: Memory layout of a 3-stream streamset

Regrettably, because we must provide a consistent memory layout for multiple kernels who could have differing *stride_length*s, we fix the bit-width of each stream element to $w \times$ *block_width*. Currently this matches the transfer rate of every I/O port whose kernel transfers multi-stream streamsets but this should be reassessed in the future.

Apart from element granularity and layout, there are a few important concepts to consider w.r.t. streamsets. Firstly is the idea of an **owned vs. unowned buffers**. A kernel who owns a buffer is responsible for managing it and buffer management is dependent on its other remaining features. The pipeline (discussed in §2.1.5) typically owns every streamset buffer except for those associated with its own I/O ports or those whose production rates are `Unknown` or explicitly marked as a `ManagedBuffers`. Currently, the only instances of `ManagedBuffers` are associated with the `MMap` and `Read` source kernels (defined in §2.2.1) whose outputs refer an OS managed buffer or an internal copy-back buffer, respectively.

The next feature we consider is **buffer type**. From the perspective of a kernel, this type changes depending on whether its owned or unowned by that particular kernel. An unowned buffer is always viewed as `External`, defined shortly. Owned buffers can be either `Linear` or `Circular`. `Linear` buffers rely on an internal copy-back mechanism to preserve memory by shuffling data back after consumption thereby overwriting consumed positions. `Circular` buffers avoid this complexity by relying on an internal modulus function to index the appropriate element positions but — as we discuss later — are significantly more complicated w.r.t. dynamic memory management. `External` buffers are treated as linear buffers but the associated kernel is not responsible for (nor permitted to perform) any copy-back operations. Depending on the `ProcessingRates` (§2.1.2) and `Attributes` (§2.1.6) applied to the associated I/O ports, the opposing end of the data-portion of an owned buffer may need to be mirrored within these regions to provide `External` buffers with a seamless view of the data. Currently any streamset linked to a I/O port with a `LookBehind` attribute will require an underflow region to be maintained. Similarly, any streamset associated with a `LookAhead` attribute or non-`Fixed` input rates will require mirroring data within an overflow. In theory, mirroring for `Circular` buffers could be effectively free with kernel-level access to the OS virtual memory manager byway of mapping a single physical memory page to multiple virtual addresses [5]. This optimization has not been explored in Parabix.

Our next topic is about the expected **throughput** of streamsets and its affect on the memory management subsystem. Because the total data processed during JIT-execution could exceed the available system memory and processing all of the data at once is not conducive to pipeline-parallel work, JIT-execution is divided into segments.

Each kernel logically processes one **segment** of data per invocation but the size each segment aka its *segment_length* is a fluid notion. Even in a fully `Fixed`-rate program, wherein we can precisely define the flow of data at JIT-compilation [61, 71], the *segment_length* of each kernel may differ. Once non-`Fixed` rates are introduced to a program, a kernel's *segment_length* may differ between successive segments. During JIT-compilation, one task of the pipeline (discussed primarily in §4.2.3) is to determine the maximum expected throughput of each kernel, thereby determining the *required_capacity* of each buffer. Unfortunately, while we can often make good worst-case predictions, we cannot determine a concrete number for *required_capacity* when one of its associated *segment_length*s is *symbolic*. If and

only if the lower bound of all of the input rates associated with a streamset always meets or exceeds the upper bound of its linked output rate do we have sufficient information to determine a concrete upper bound of its *required_capacity*. Such buffers are instantiated as **static buffers**, named due to their capacity being fixed at JIT-compilation. Other buffers are designated as **dynamic buffers** since they may need to be dynamically expanded to prevent a program from deadlocking [94].

Dynamic expansion introduces a significant complexity: not only do we have to correctly move all unprocessed data into new buffer but we must also consider the impact of parallel execution. Expanding `Linear` buffers is straightforward but the unprocessed data within `Circular` buffers may be split into two disjoint regions, which must be copied into positions that correctly reflects its new modulus function. To accommodate parallel accesses during expansion, the pipeline utilizes a *double buffering* strategy. When expanding a buffer, a new buffer is allocated but the existing buffer is not immediately freed because a producer of a streamset cannot be certain which of the streamset's consumers are still actively reading from that buffer without using some form of synchronization in a multi-threaded environment. Rather than slowing down the program's critical path to handle a rare event, each thread independently stores a list of pending free-able buffers. A natural property of our pipeline is that it is always safe for a producer to free a buffer added to the list during its previous segment on the same thread. We describe the pipeline in more detail in §2.1.5.

Because the generated `DoSegment` logic can be arbitrarily complex and the conjunction of all the JIT-ed `DoSegment` instructions is often too large to fit within the L1 I-Cache, we amortize the cost of fetching them during JIT-execution by transmitting large sequences of data through each I/O port per segment. Consequently, the *segment_length* is often set to a value that entails the *required_capacity* is quite large. since threads typically process buffers of 4–64 *KB* per segment. As discussed later in §3.1.2, modelling dataflow programs with large shared buffers is in stark contrast with known dataflow models.

2.1.4 Scalars

A scalar is a single w-bit value. Four types of scalars in the Parabix framework: input, output, internal and temporary, each of which serve an explicit purpose, detailed shortly.

At edit-time, kernels can be configured by passing arguments to them during instantiation. Generally speaking, passing `C++` constructor arguments implies we will JIT-compile a different instance of the kernel code for every configuration of concrete settings. For example, were we to pass a file descriptor as a concrete argument to a kernel, that kernel would be JIT-compiled uniquely for every observable value of file descriptor witnessed by the program at run-time. Obviously, given the caching system (discussed in §2.1.7), this would be grossly inefficient for any program other than those restricted to reading input via `stdin`; thus such parameters are typically passed *symbolically* into kernels during JIT-execution.

To provide this functionality, kernels can be associated with input and output scalar objects. Similar to streamsets, any number of kernels can read the same input scalar but only one may write to an output scalar. During JIT-execution, input scalars are passed into the kernel `DoInit` method as concretized values via the pipeline (discussed in §2.1.5). These parameters are eventually written a protected region of the associated kernel state object for use during the `DoSegment` and `DoFinalize` stages. Internal, output and temporary scalars, on the other hand, are mutable fields and may be freely modified by a kernel within any of its methods. Both internal and output scalars are stored as fields within the kernel state. Internal scalars can be marked as either shared or thread-local (depending on their intended use) but output scalars are always shared. Temporary scalars, however, are zero-initialized fields that refer to a function-scoped stack-allocated variables in the current method.

Both the `Pipeline` (§2.1.5) and `Pablo` kernels (§2.3) rely heavily on temporary scalars to reduce their memory footprint when possible (with the assumption that LLVM's `mem2reg` pass can promote them) but retain a common API for when such optimizations are impossible. The `Pipeline` is similarly heavily dependent on thread-local internal scalars to maintain some critical thread-specific state. To support nested pipelines, this functionality was extended to all kernels. Another important feature for the `Pipeline` is the notion of **scalar groups**. The `Pipeline` state contains shared state information for every invoked kernel. Tightly packing kernel state could lead to *false sharing*[3] when accessing it during a multi-threaded execution. Scalar groups pack internal scalars into cache-line aligned sub-structs by automatically inserting enough padding between groups to prevent the state stored within any group from sharing the same cache line as another group. This increases its memory footprint by up to [(cache-line size $-$ 1) $\times$ num of groups] bytes.

Assuming an output scalar is an output of the `Pipeline`, it will be gathered by the `Pipeline` and returned to the `C++` program after calling the kernel `DoFinalize` method. Currently Parabix does not support mapping an internal or output scalar of one kernel to an input scalar of another but this would be a useful feature to consider in the near future (e.g., a `base64` transcoder could trivially calculate the output file size based on the input file size; this information could be passed from a file input kernel to a file output kernel via an internal scalar.) Mapping an output scalar to an input would be significantly more difficult because it necessitates a multi-stage execution. This too is an obvious avenue

[3]Modern CPUs contain relatively small caches that ideally retain commonly accessed memory in a location that is physically closer than main system memory, reducing memory latency. Memory blocks are deterministically mapped to cache lines and the cache-line size, typically 64 or 128 bytes, is the smallest quantum of memory that can be stored in the cache. False sharing can occur when two cores simultaneously access differing bytes within the same cache line. When one of the accesses is a write, the cache coherence protocol will automatically invalidates the line in the other core(s) cache. False sharing occurs when a core is forced to reload a cache-line from a higher-level memory storage even though the desired byte(s) were unmodified by the off-core write.

of future development that can be supported by allowing `DoInit` and/or `DoFinalize` to construct and call their own prelude/postlude pipeline(s).

2.1.5 Pipeline

The `Pipeline` kernel — named for the form of parallelization it provides — is the backbone of every Parabix program; it is responsible for handling multi-threaded synchronization, the majority of memory management concerns and invoking kernels in some logical order. After compiling the kernels, the `JITDriver` (§2.1.11) builds a `Pipeline` to encapsulate and call each kernel during JIT-execution. JIT-compilation ends after the `Pipeline` is compiled and JIT-execution begins once the `Pipeline`'s "main" method is started by the `C++` program. We begin this section with a brief discussion of pipeline parallelism, then we explain the responsibilities and functionality of the `Pipeline` kernel during each life-cycle phase and then describe the thread synchronization mechanism used by the framework. We conclude with a comment about alternate forms of parallelism that could incorporated into the `Pipeline` for future work.

Pipeline paralleism

Pipelining is a technique wherein a program is decomposed into discrete stages that communicate with each other via buffers. Conceptually, these buffers can be viewed as FIFO queues. Each stage pulls data from its input queue(s) and pushes data into its output queues, which in turn are input queues of some subsequent stage(s) [15]. Parallelizing such programs is trivial since each stage is ideally suited for thread-parallelism.

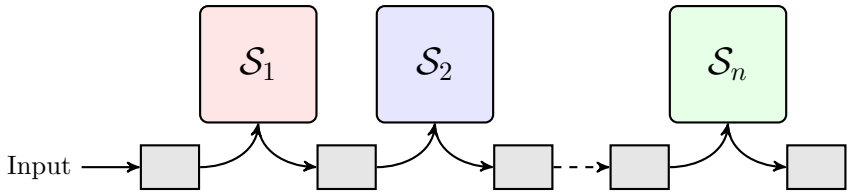


Figure 2.3: Abstract linear pipeline with n stages (Adapted from [15])

Pipelined programs are either linear and non-linear, depending on whether the digraph generated by the stage and buffer relationships is acyclic or cyclic, respectively. For example, Figure 2.3 shows a simple linear pipeline of n SISO stages with buffers connecting each. Because of the design of the both the current and foreseeable Parabix programs, we only consider the **linear-pipeline** model within this dissertation. Generally speaking, linear-pipeline programs fit into one of two categories, fixed-code or fixed-data [15] but recent work by Mastoras explored a hybrid model [78]. With the fixed-code approach, stages are partitioned into threads, pinning the code for those stages to a particular processor at the cost of transferring the I/O data between them. In the fixed-data model, every thread executes the same code (i.e., the entire program) but the data that it transfers between

stages is statically mapped to a thread. Instead of transferring data between threads, systems using a fixed-data model transfers stage-state information between them. Maximizing throughput in a multi-threaded fixed-data scenario requires that each thread executes the stages within a loop, devoting only a small quantum of processing time per iteration to each stage before moving to the next. However, because of its inherit load balancing, the fixed-data model avoid many of the load-balancing requirements to maximize throughput in fixed-code programs.

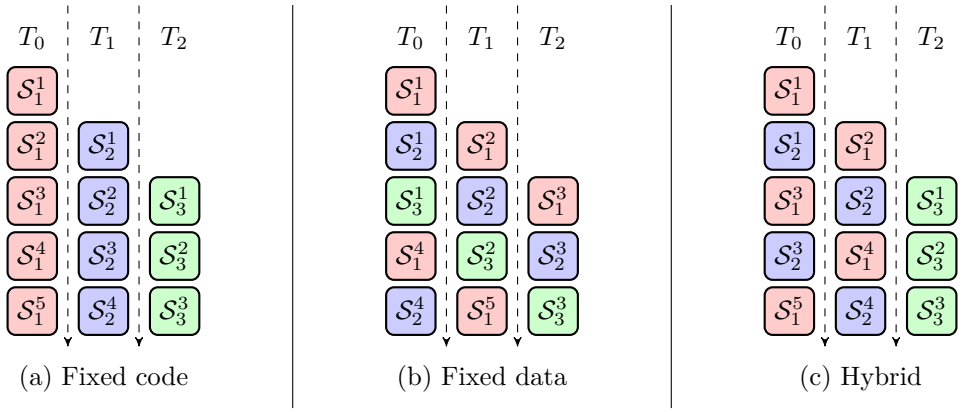


Figure 2.4: Pipeline parallelism models (Adapted from [79])

With the hybrid approach, each program is initially viewed as s fixed-data system. A program stage can be isolated onto a "fixed-code" thread and executed in parallel with the rest of pipelined system. When the isolated stage(s) dominate the running time of the program, hybrid systems can improve throughput by dedicating computation resources to the most expensive components of the program [78]. We briefly discuss our implementation of hybrid threads in §4.5 and evaluate it in §5.6.

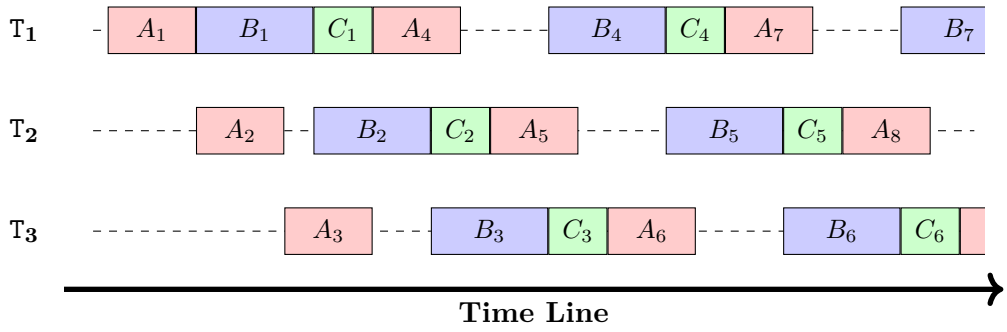


Figure 2.5: Fixed-data pipeline execution with 3 threads (Adapted from [75].)

Parabix primarily uses a *relaxed fixed-data model* but supports the hybrid model (§4.5). Figure 2.5 provides a slightly more detailed view of this model than Figure 2.4b. Although

several methods of multi-core acceleration were originally considered for Parabix, this variation of the fixed-data model was found to be the most performant and scalable on average [75]. To contend with non-`Fixed` rate I/O, this relaxation permitted data to migrate between threads but forces kernels to aggressively consume input data to mimic a fixed-data execution pattern. As implied, with this model the code for each thread is identical but the code for kernel K_i at time j may only be executed after the code for K_i at time $j-1$ has completed. Consider the example with 3 kernels A, B and C and 3 threads T_1, T_2 and T_3 in Figure 2.5. Thread T_1 begins executing kernel A at time 1, denoted in the figure as A_1; T_2 must wait for A_1 to complete before it may process A_2 but T_1 is free to begin executing B_1 immediately after releasing the synchronization lock for A_1. Similarly, T_3 waits on A_2 and T_2 must wait until both B_1 and A_2 are complete before invoking B_2.

JIT execution

JIT-execution of a `Pipeline` kernel is divided into three distinct phases: ① initialization, ② segment-processing and ③ finalization. Each phase is associated with one of the overridable kernel methods. During *initialization*, each kernel `DoInit` function is called once. Unlike the prework function of StreamIt [109], Parabix kernels cannot access their I/O ports during initialization but can make any other internal configurations or allocations. Input scalars, such as file descriptors, from the outside program can be passed into the kernels during this phase (and only this phase) via the pipeline `main` function, defined later in this subsection.

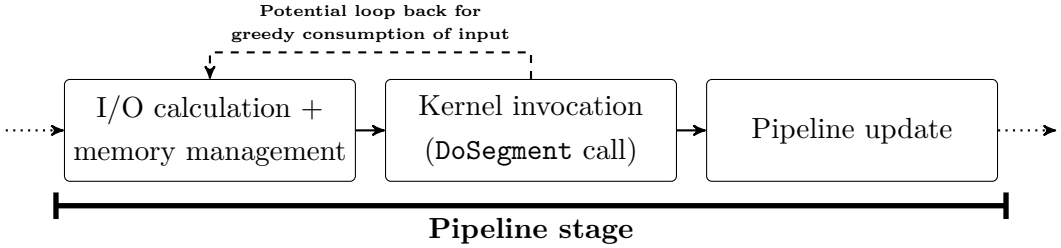


Figure 2.6: Abstract pipeline stage definition

The *segment-processing* stage is the work phase of the program. Here the `Pipeline` invokes the `DoSegment` logic of each of its contained kernels. Each invocation is encapsulated within a pipeline stage, which we abstractly depict in Figure 2.6. Conceptually, each pipeline loop iteration processes a single segment of information. During JIT-execution, each kernel `DoSegment` function is logically invoked once per segment but because non-countable input cannot be predicted a priori and circular buffers can be unaligned w.r.t. their modulus position, fully processing a segment may actually require multiple `DoSegment` calls. Although this does not typically affect a kernel programmer, they should always obey the accessible/writable I/O restrictions as dictated by the pipeline at run-time.

The total duration of the pipeline segment-processing phase (i.e., number of segments processed by each kernel) is dependent on the input data, including any data produced by the source kernel(s), described in §2.2. Although the amount of data that each kernel processes per segment can vary, generally speaking each kernel operates on the data that was produced by the prior kernels within the same period.

As the name implies, the *finalize* phase acts much like a typical `C++` destructor. While the actual behaviour of the `DoFinalize` function varies depending on the program, within this phase kernel output *scalars* are gathered by the pipeline and returned to the `C++` program. This is the only phase in which the `C++` world can view any streamset data or values from the kernel state objects. Any other artifacts obtained by the pipeline execution must be outputted by specially marked `SideEffecting` kernels. Note here that such kernels are still prohibited from modifying the state of any kernel within the pipeline. Side effecting implies that they modify the state of the *outside world* in a way that cannot be observed via the I/O relationships (e.g., writing to `stdout`.) It is the programmer responsibility to ensure side-effecting kernels cannot halt or otherwise conflict with the execution of any pipeline.

Thread synchronization

Thread synchronization prevents two threads from simultaneously executing the same protected region of code. Our chosen method is a cooperative *ticket-lock* mechanism. For each synchronization lock $\mathcal{L}$, the pipeline contains a 64-bit integer $\mathcal{L}_T$ within its (non-thread-local) shared state. The value of $\mathcal{L}_T$ always refers to the logical segment number of $\mathcal{L}$'s current ticket holder. At the start of each linear-pipeline loop iteration, the pipeline acquires the next available segment number S. To enter $\mathcal{L}$'s protected region, the thread must acquire $\mathcal{L}$, which the pipeline does by busy waiting until the thread executing segment $S-1$ releases $\mathcal{L}$ by writing S to $\mathcal{L}_T$. Similarly, after executing the region's code, the current thread will write $S+1$ to $\mathcal{L}_T$ to release $\mathcal{L}$ for the next thread.

As Figure 2.6 shows, each pipeline stage consists of three phases with the kernel invocation (i.e., the calling of the kernel `DoSegment` logic) being the critical one from the programmer's perspective. With stateful kernels the protected region encompasses the entire pipeline stage but for the subset of *state-free* kernels we can permit concurrent executions of the `DoSegment` code so long as we guarantee that every other interaction with the pipeline and memory management subsystems occurs in a thread-safe manner. Each state-free kernel has a pair of pre- and post-invocation locks. Like their names suggest, upon stage entry, the pre-invocation lock is acquired and released immediately before kernel invocation. The post-invocation is acquired immediately after the `DoSegment` returns and released when exiting the stage. However, if any linear copybacks or circular buffer expansions are required, the pipeline acquires the post-invocation lock prior to the `memcpy` to ensure any copied data reflects the state each streamset buffer would be in during a serial execution.

Being stateful or state-free is more than the inclusion or absence of internal scalars. The I/O of a stateless kernel must all be a non-`Deferred` countable rates[4] and the kernel cannot self-terminate since the deferred position and termination signal are implicit forms of program state. Additionally, `SideEffecting` kernels are assumed to have an implicit dependence on the outside `C++` program or OS, which prevents them from safely exploiting this optimization. An `StateFree` attribute can override the internal scalars and `SideEffecting` restrictions but will report an error if asked to override other limitations. Finally, `InternallySynchronized` kernels are considered effectively "state-free" from the perspective of the pipeline but place the responsibility for correctly updating the I/O state in a thread-safe manner on the invoked kernel.

Neither mutexes nor hybrid spin-then-park locks were explored in this dissertation. Preliminary investigation shows Parabix might benefit from using hybrid locks for `mmap` and `read` source kernels (discussed in §2.2.1) because their busy-wait time can account for a significant portion of the total run-time whenever their input data is not already within the OS page/buffer cache on some architectures. This assumes, however, that a program is heavily I/O-bound — as opposed to compute-bound — which is not the expected use-case of this framework. Similarly, programs whose run-time is dominated by a small subset of kernels may also benefit from these alternative locking mechanisms but even with an optimal locking strategy, such programs would still be greatly limited w.r.t. their maximum speed-up. We leave exploration of this topic to future work.

Comment on alternate forms of parallelism not considered in this dissertation

In this dissertation, we primarily consider linear-pipeline parallelism (of predominately SIMD-based kernels) but the maximum acceleration of any linear-pipeline program is limited to the total run-time divided by the cost of its most time-consuming stage. Other forms of parallelism could theoretically could take advantage of more cores than pipeline parallelism permits.

This framework supports a limited form of **data-parallelism** for state-free kernels but more complex forms of multi-core acceleration are possible for stateful kernels. For example, any stateful kernel that otherwise adhere to the I/O and attribute constraints but whose processing can be divided into regions with trivially-detectable demarcation points can be treated as state-free by arbitrarily executing the preceding region of every segment. Whether this falls under data-parallelism or speculation is dependent on whether the region is statically or probabilistically bounded in length. True **speculation** (i.e., the ability to conditionally execute the $k^{\texttt{th}}$ iteration of a stage before the thread executing

[4]Formally defined in §2.1.2, a countable processing rate is one in which the pipeline can determine how many items will be transferred through the I/O ports prior to the `DoSegment` call. A `Deferred` rate acts like a countable rate from the when the pipeline determines the threshold for "new data" but permits the programmer to mark how many items have been fully processed/produced.

the $k-1^{\texttt{th}}$ iteration of the stage has finished) is also possible but comes with significant caveats. Speculation requires that a program has some form of detection, rollback and recovery mechanism s.t. upon the $k-1^{\texttt{th}}$ iteration's completion, a controller can determine whether the assumed state of the speculative execution of the next stage was correct (w.r.t. producing the correct output) and fixing the output in the cases wherein it was not prior to generating any observable results to the end user. Generally, speculation can only improve performance if the set of likely states is small enough (and/or predictable enough) that the total detection and rollback time does not exceed the synchronization delay.

With that being said, speculation has been applied to many semi-structured text-parsing problems, notably probability-based CSV parsing [43]. This form of speculation is a product of program design rather than framework compilation strategies. A more interesting case was explored by Qiu et al. on carry-flag state prediction of `Pablo` (and similar) programs [98][5]. They reported a $10.4-27.6\times$ speedup with a 64-core machine compared to a single-threaded program. Although their speedup falls way short of our speedup goal, with only $3-4$ logical stages, it vastly exceeds what is achievable with pure pipeline parallelism. However, the examples chosen by Qiu et al. were all of simple `Fixed`-rate programs in which they reported that serialization of output was a major limiting factor. The most complex examples were REs for `icgrep` that required speculating the value of 19 to 36 bits of information. Although correctly predicting 36-bits is impressive, this framework is intended to support a broad class of applications with state that is too complex to efficiently reason about using FSM-based speculation techniques. For example, REs matching tends to be very consistent (in that matches tend to be very sparse and occur in short dense bursts) so simply reusing the last known carry state will likely to lead to the correct observable behaviour in the majority of the cases — but this cannot be said of general program state, which can include 64-bit integers, hash tables, and partially-computed pending output data.

Finally, for some types of programs, **task-parallelism** is a natural form of acceleration to consider. For example, `icgrep` supports task-parallelism in the form of a work queue when processing multiple files. The available threads are evenly divided amongst the task groups and each group independently executes a `Pipeline` using the available resources. Every group pulls the next available file to process from a shared queue until the queue is emptied. At a high level, this design cannot be improved upon in `icgrep` but it does overlook the overhead involved in memory allocation and the cost of thread startup and teardown that occurs every time a `Pipeline`'s main function is executed. Moreover, it also ignores the potentially non-cooperative nature of these tasks since they do not coordinate their file I/O, potentially increasing bus traffic and decreasing throughput of all of the tasks. A single `Pipeline` could easily replicate the task-based nature of this with differing kernel state objects and a clever use of synchronization to coordinate file I/O but replicating

[5]Code and data artifacts for [98] are available at `https://zenodo.org/record/3610556`

the work queue functionality is far more difficult. At minimum, even with a single task group, supporting task-parallelism requires that upon pipeline termination (either from an EOF or error condition) that the `Pipeline` effectively restarts by zero-ing the pipeline and kernel state before instructing the source kernel to open the next file in the queue. With multiple task groups, the pipeline must accommodate the fact these restarts will happen at differing points for each group but the impact of the restarts must be localized to the threads belonging to that group.

2.1.6 Attributes

As their name implies, **attributes** provides a way to associate kernels and I/O ports with properties that distinguish those entities from the normal case. This section is *intended as a reference guide and can be safely skipped* but for the interested reader, we list the set of attributes currently supported by the Parabix framework below:

Input port attributes

- `LookAhead:` a `LookAhead(`k`)` attribute on an input stream set **I** declares that the kernel looks k positions ahead when processing each item of **I**. When determining a kernel's *segment_length*, the pipeline ensures that the kernel can safely read $\ell = \lceil k/stride\_length \rceil$ strides ahead of current segment end. Consequently, `LookAhead` requires an *overflow* region in the associated streamset buffer. More importantly, `LookAhead` greatly complicates the flow of data through the pipeline. The main issue can be seen in Figure 2.7. Suppose kernel $\mathcal{K}$ has a `LookAhead` dependency on streamset **I**. For $\mathcal{K}$ to process stride i, denoted by $\mathcal{K}_i$, it must read the input **I** generated for stride $i + \ell$. For this data to exist, the pipeline must initially withhold at least ℓ strides of data from **I**. This effect is compounded when multiple kernels with `LookAhead` dependencies are chained together. Although this is a one-time deferral, it causes a "misalignment" in the data access pattern of `Circular` buffers.

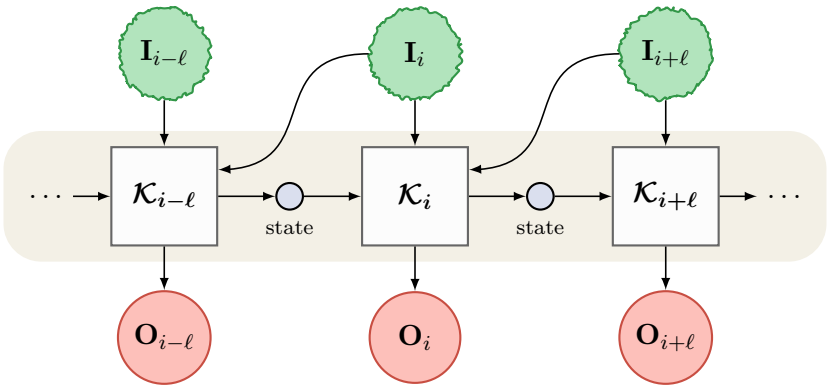


Figure 2.7: Kernel Lookahead input dependency

- `LookBehind:` a `LookBehind(`k`)` attribute ensures that k (or more) positions prior to the current processed position of the associated input port must be stored in contiguous

memory w.r.t. that position. If the current position is the $< k^{\texttt{th}}$ position of that input, the "items" behind it will be zeroed. Unlike `LookAhead`, `LookBehind` will not introduce any segment delays into the pipeline but will force the pipeline to allocate an *underflow* region in the associated streamset buffer during initialization.

• **`Principal:`** a single input port can be declared as the principal input for a kernel. If the principal input is `Fixed`, the available item count of the streamset overrides the available item counts of any other `Fixed`-rate inputs during final-mode processing, implicitly zero-extending/truncating them to match. This feature differs from the `ZeroExtended` attribute in that it will only extends each stream up to a single *stride_length* but avoids the unnecessary `malloc` potentially associated with `ZeroExtended`-ed streamsets. We discuss this more in §2.1.2.

• **`ZeroExtended:`** if the available item count of an input port is less than some other input streamset(s), the streamset will be zero-extended to the length of the larger streamset. If this option is not set and the kernel does not have a `MustExplicitlyTerminate` attribute, the kernel will enter final-mode processing once any input streamset has been exhausted. We discuss this more in §2.1.2.

NOTE: `ZeroExtended`-ed streams are not considered by the pipeline when ascertaining whether a kernel is entering its final segment. At least one input stream must not be `ZeroExtended`-ed and a port may not have both `Principal` and `ZeroExtend` attributes.

Output port attributes

• **`Delayed:`** when applied to an output port, a `Delayed(`k`)` attribute indicates k produced items must be withheld from processing by other kernels until the producing kernel enters its final segment. Assuming an output streamset currently contains n items, only $n - k$ items can be safely read prior to the producer's termination and n otherwise.

• **`EmptyWriteOverflow:`** to avoid branching, some kernels (e.g., `StreamCompress`) may write an output buffer despite having no items to produce. When such writes occur after the end of a streamset, this can cause a segfault if the pipeline has not allocated an overflow region for it to write into. This attribute will inform the pipeline an overflow is needed but will not actually store useful data.

• **`ManagedBuffer:`** In [75], every kernel was responsible for resizing its dynamically-sizable output buffers. Recently, the responsibility of managing non-`Unknown` rate output streamsets was absorbed by the pipeline. Some kernels, however, may still want to manage their memory internally. For example, source kernels (discussed in §2.2.1) produce data at countable rates but `MMapSourceKernels` returns memory directly from the OS; `ReadSourceKernels`, on the other hand, pull data from `stdin` (or another file descriptor) but relies on an internal copy-back buffer mechanism to return contiguous data to the pipeline. To treat both kernels as belonging to the same family, both output ports are marked as managed buffers.

• **`RoundUpTo:`** Primarily intended for countable rate outputs, A `RoundUpTo(`k`)` attribute indicates the final produced item count of an output streamset must be rounded up to the nearest multiple of k. This is currently used by the Parabix `base64` encoder to accommodate its padding rules.

Any port attributes

• **`Add:`** `Add(`k`)` indicates k items will be appended to the total transferred item count of a countable port. This attribute is primarily used to model the fact that many `Pablo` instructions (discussed in §2.3) tend to leave "cursors" k positions to the right of the fields of interest. This can have a cascading effect when chains of `Pablo` kernels exist in the pipeline that must be reasoned about to prevent memory corruption. When applied to an output, the available item count of the associated streamset will reflect the k added items; when applied to an input, only that particular kernel will see k additional items.

• **`BlockSize:`** both the pipeline and kernels assume that each stream of a streamset is divided into linear sequences of *stride_length* items. A `BlockSize(`k`)` attribute informs both that the streamset is actually divided into (*stride_length*/k) items and each "stream" actually contains k items of the first stream followed by k items of the second stream and so on. The notion of transferred item count changes to suit. Suppose a port without a `BlockSize` attribute report that it has transferred up to the i^{th} item, indicated by `*`, of a 4 element streamset. The streamset layout would be:

```
... | AAAAAAAA AAAAAAAA AA*..... ........ | ...
... | BBBBBBBB BBBBBBBB BB*..... ........ | ...
... | CCCCCCCC CCCCCCCC CC*..... ........ | ...
... | DDDDDDDD DDDDDDDD DD*..... ........ | ...
```

However, if (*stride_length*/k) is 4, the same i^{th} position above is actually:

```
... | AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD | ...
... | AAAAAAAA BBBBBBBB CCCCCCCC DDDDDDDD | ...
... | AA*..... BB*..... CC*..... DD*..... | ...
... | ........ ........ ........ ........ | ...
```

• **`Deferred:`** normally the transferred item count of `Fixed` rate port is calculable based by the number of strides already executed by the kernel. However, some ports behave like *countable* ports in that they always transfer a set amount of data but the kernel releases it at unpredictable rates. For example, the `ScanMatchKernel` — a core component of `icgrep` — forwards each line of text that matches the RE to the `C++` program but it cannot refute one exists until it sees the next newline or `EOF` demarcation. However, the processing frontier of what it considers new input byte-data is equivalent to a `Fixed` rate so rather than declaring that port as having a `Bounded` rate, marking it as a `Deferred` and `Fixed` enforces a much

stronger throughput guarantee. `Deferred` attributes can be applied to any countable rate port but its greatest utility is found when combined with `Fixed` rates.

When the `Deferred` attribute is applied to an input port, the pipeline simply refers to the deferred number when calculating how many items have been consumed from the streamset but when applied to an output port, the `Deferred` attribute has a very different meaning. To the pipeline, streamsets linked to a deferred output port must always have sufficient space to satisfy the upper bound of its base processing rate as if it always produced the maximum number of items per stride, regardless of its current deferred value. Such ports risk requiring unbounded memory when data is withheld for the lifetime of the program and should be used with caution.

- **`Linear:`** indicates that an input streamset must be backed by a contiguous memory buffer or is an output `ManagedBuffer` that promises to be linearly accessible. Since streamsets can represent either linear (copy-back) buffers or circular buffers, this can greatly impact the complexity of the pipeline.

- **`Truncate:`** opposite of `Add`(k); removes k from the available item count.

NOTE: supports only the removal of "added" items.

Kernel attributes

- **`CanTerminateEarly:`** indicates a kernel may terminate before its input is exhausted. A terminated kernel cannot be invoked after its termination signal is received.

- **`MustExplicitlyTerminate:`** states a kernel finished only after the programmer has explicitly terminates it and that the kernel must be called at least once per segment even if no new input is available for it to process.

- **`MayFatallyTerminate:`** allows a kernel to initiate the termination of the pipeline by calling the `KernelBuilder.setFatalTerminationSignal()` function. The caveat here is that function neither terminates the kernel nor will the pipeline immediately respond to the signal. Instead, the pipeline checks whether any kernels have fatally terminated at the end of its current `DoSegment` loop iteration. Due to multi-threading, discussed in §2.1.5, up to $num\_of\_threads - 1$ additional segments could be executed by the pipeline. Note, however, no kernel will be invoked after termination.

- **`Family:`** indicates this kernel belongs to a family of kernels in which every member of the family exactly matches this kernel's I/O and attribute signature (including the `Family` attribute). We discuss kernel families in §2.1.7.

- **`InfrequentlyUsed:`** by default, every kernel is JIT-compiled using `-O2` optimization level. Some kernels may be complex enough that when comparing the JIT-execution time of the code compiled at `-O0` and `-O2` level, the additional JIT-compilation time could exceed the benefit of a typical program execution run.

Any kernel marked as `InfrequentlyUsed` is instead compiled at `-O0` by default. This can be overridden with the `backend-optimization-level` command line argument; note that the optimization level is incorporated into the kernel signature and the caching infrastructure will not look for matching kernels compiled at differing optimization levels.

• **`InternallySynchronized`:** the synchronization mechanism used within the framework prevents two threads from simultaneously invoking the same stateful kernel. Since `Pipelines` obviously contain their own synchronization guards around any of their internal kernel invocations, this needlessly serializes any invocation of a nested `Pipeline`. To prevent this performance degradation, the `PipelineCompiler` treats `InternallySynchronized` kernels as state-free and substitutes its synchronization guards accordingly (§2.1.5).

• **`SideEffecting`:** reports to the pipeline that the kernel interacts with the outside program in a way that cannot be observed via its I/O relationships. A pipeline will not discard a `SideEffecting` kernel that produces no outputs nor will it assume that two identical side effecting kernels with the same inputs produce an equivalent output.

• **`StateFree`:** discussed in §2.1.5, any kernel with strictly non-`Deferred` countable I/O rates and no internal or output scalars nor any termination or `SideEffecting` attributes is considered state-free. This attribute will override the limitation on internal state and the `SideEffecting` attribute but will place the responsibility of thread-safe accesses upon the programmer.

• **`IsolatedOnHybridThread`:** every Parabix program is assumed to be a multi-threaded linear-pipeline program. Typically, each thread executes the same sequence of kernels, which is known as a *fixed-data* linear-pipeline. Occasionally, it may be possible to isolate a set of kernels to run on an independent thread to improve throughput, essentially executing *hybrid* linear-pipeline. This attribute marks a particular kernel for hybrid execution. We discuss the various forms of linear-pipeline parallelism in §2.1.5.

2.1.7 Object cache

No matter how simple a Parabix program is, JIT-compilation is an expensive component of the Parabix framework. Ideally, JIT-execution will vastly dominate the run-time phase but without caching, this is only true when processing very large files. Although outdated w.r.t. the current framework, Dale Denis's study found JIT-compilation in `icgrep` represented $\geq 20\%$ of the total run-time of `icgrep` when processing files of $\leq 75MB$ and only became negligible at $300MB$ or more [34]. Both the complexity of the framework and the expected JIT-compilation time of later versions of LLVM have increased since the time of this study.

To mitigate this trend, the Parabix framework employs an extensive caching system. Prior to compiling any kernel (including the pipeline), the framework inspects the `.cache` directory (or OS-specific alternative) for a precompiled version of the desired kernel, using one instead whenever possible. This is realized by means of the `ParabixObjectCache`.

Overall design

The `ParabixObjectCache` is an extension of the `llvm::ObjectCache` and directly links into the LLVM compilation tool chain. Before compiling a `Module`, LLVM calls upon the `ParabixObjectCache` to check whether a matching precompiled `Module` exists. The file name in which the `ParabixObjectCache` checks the `.cache` for is composed of both the kernel name and an embedded date/time code indicating when that particular executable was compiled. The former identifies the kernel in question the latter ensures the cached instance was compiled with the same version of the executable that is requesting to run it. To minimize the start-up and JIT-compilation time, we place a hard constraint on when a kernel can be assigned a name. Specifically, *a kernel's name is determined solely at instantiation* (i.e., it is fixed immediately after returning from the constructor.) Modifying a kernel name after instantiation is undefined behaviour and will be ignored by the caching infrastructure.

Included in the `.cache` for each kernel is a `.o` object file and a `.kernel` LLVM metadata file. As its name implies, the `.o` file contains the final optimized machine code; the **`.kernel`** file is a LLVM bitcode file consisting of ① a kernel signature and ② its generated state object type and function prototypes.

Even though kernels ought to be distinctly named, not all names can be used as a valid filename. For example, an `icgrep` RE kernel name is derived from the raw RE text provided by the command-line; an RE can be arbitrarily long but the maximum filename length on Linux is 255 bytes. For such kernels, the `ParabixObjectCache` substitutes the kernel name for its hex-encoded `SHA-1` hash-code when generating its cache name. Although this hash-code is probabilistically unique, the original name is encoded within the `.kernel` as its **signature** and checked by the `ParabixObjectCache` prior to accepting the `.o` file as a match. A hash collision will result in the existing file pair being discarded and JIT-compilation continuing as if no file pair was found.

Perhaps the most overlooked piece of metadata is the **function prototype**. Although a "type-less" prototype for each of the three kernel methods can be directly determined by the number of streamset and scalar arguments for each kernel, the exact size and shape of each *kernel state object* differs wildly. For example, a `PabloKernel` state object contains numerous carry-bit fields necessary to facilitate a segment-based parsing model. We defined what this entails in §2.3.4 but for now it suffices to say that obtaining this means generating, optimizing and partially compiling the `Pablo` IR to LLVM IR. The state object for every kernel must be allocated during JIT-execution time. Compilation of complex `Pablo` kernels, such as those generated by `editd` with a large edit-distance setting, can take an excessive amount of time. Although still relatively small compared to the LLVM compilation overhead, it can take several seconds or even minutes with very large settings. We amortize this cost by recording the state object type in the `.kernel` metadata.

Cache janitor

Unfortunately, we cannot rely on OS to manage the `.cache` and as time passes, programs like `icgrep` could cache enough files that it becomes problematic. Removal of "stale" files (i.e., those who have not been used within a set time period) is necessary to maintain a healthy file system. However, in systems where `icgrep` is frequently used, the time it takes to iterate through the `.cache` and check the last accessed time of each file is significant — enough so to dominate the run-time cost when checking small files. Earlier attempts at reducing this included randomly testing only k files but this took computation resources away from the main application and often aborted before finding k due to the main program exit. Currently, the `ParabixObjectCache` spawns a lightweight **cache-cleanup daemon** (when one is not already active on the system) but this requires bundling the program with the daemon executable. Simplifying this process is an avenue for future development.

Kernel families

Originally, a pipeline function was constructed and JIT-compiled anew every time a Parabix program entered its JIT-compilation phase; its sole responsibilities were synchronization and correctly forwarding the appropriate streamset I/O state to each kernel. All of the burdens for buffer management, kernel sequencing, and dataflow shaping was placed upon the programmer [75]. Automating this analysis drastically increased the JIT-compilation time of the program. We mitigate this using caching but understanding how requires appreciating why the original design decision was made.

Consider the following: `icgrep` is a standalone replacement for the `grep` command line tool; it accepts an input RE and file name(s) from the user and returns some information about the matching lines within the files. Every distinct RE corresponds to a unique RE kernel object. To directly call each kernel's `DoSegment` method, `icgrep` required a unique pipeline function to be compiled for each RE kernel. Ensuring the pipeline was as cheap as possible to compile minimized `icgrep`'s JIT-compilation overhead.

To remedy this problem, `icgrep` pipelines were given the ability to invoke RE kernels by *function pointer*; thus the same `icgrep` pipeline can handle any RE. Support for this was generalized as the notion of **kernel families**. Any kernels with identical I/O characteristics and attributes can belong to the same kernel family but to minimize the number of cache misses incurred by calling kernels by function pointer, only the kernels that are specially marked as belonging to some family are considered during JIT-compilation of the `Pipeline`.

Correctly instantiating a `PipelineKernel` with a family of kernels can be burdensome. The `JITDriver` alleviates the programmer of this task by compiling a `Pipeline` "`main`" function every time JIT-compilation is started. This **`main`** function is a straight-line LLVM function intended to provide a single "one-line" function call to a programmer that enables them to repeatedly start, execute, and tear down a compiled pipeline at run-time.

Pipelines without any family kernels avoid dynamic compilation. Instead we embed the `main` function directly into the cached pipeline and link to it at the start of JIT-execution time. At edit-time, invoking this function requires passing in the arguments expected by the `PipelineKernel`; the order of these parameters mirrors the order of the streamsets and input scalar values for the pipeline itself. Output scalars are currently converted into an similarly-ordered aggregate `struct` object and returned by the `main` function. Although pipelines can write to external output streamsets, as evident in `editd`, the memory management for external streamsets is currently left to the programmer — but this task could be folded into the pipeline's responsibilities. Care must be taken to ensure this does not conflict with the memory management of nested pipeline I/O, currently utilized in recursive `icgrep` searches. Both the inclusion of external streamset memory management and the ability to access output scalars by name are avenues of future work.

2.1.8 Kernel builder

As stated previously, each `DoInit`, `DoSegment` and `DoFinalize` method generates LLVM-IR during the JIT-compilation phase. Even though code for these functions can be embedded directly into the program [34], exploiting JIT capabilities requires us to generate IR during JIT-compilation. To facilitate this, Parabix provides an extended `llvm::IRBuilder`, referred to as the `KernelBuilder`. The basic `IRBuilder` class included with the LLVM library provides a common API for creating and inserting LLVM-IR instructions into `BasicBlock` objects. A `BasicBlock` is a container for LLVM instructions that will eventually be transformed into a straight-line code sequence. Each `BasicBlock` ends with precisely one terminator instruction, which can be a conditional branch, unconditional branch or return statement. For more information, we refer the reader to Ch. 3 of [69]. To the programmer, the `KernelBuilder` is a singular entity but as Figure 2.8 shows, is actually just one class in a complex architecture. In this subsection, we describe its major components.

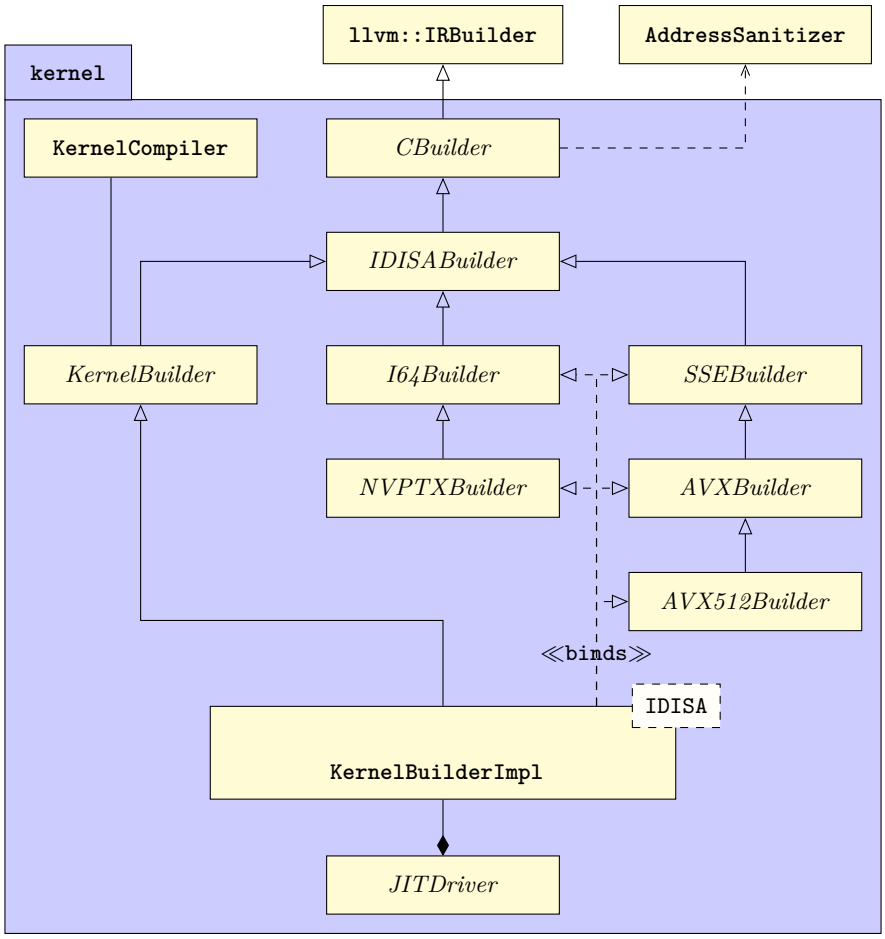


Figure 2.8: Simplified kernel builder UML class diagram

CBuilder

Shown in Figure 2.8, Parabix first extends the `IRBuilder` with the **`CBuilder`** by including common `C` functions, such as `malloc`, `memcpy`, `free` and their associated aligned variations, `pthread_create`, `mmap` and `fwrite`, amongst many others.

A key feature of the `CBuilder` is its support for **dynamic assertions**, enabled via the command line by a `-EnableAsserts` parameter. An assert, like the `C/C++` macro, reports a message that often includes a stack trace indicating where an assertion triggered whenever its expression fails to hold. A stack trace is a valuable piece of debugging information but a standard stack trace generated at JIT-execution is nearly useless. To improve the utility of assertions, the `CreateAssert` method of the `CBuilder` records the return addresses from the current call stack (w.r.t. that assertion) during JIT-compilation and reports that along with a custom message by invoking `addr2line` (or a similar program) at JIT-execution. `CreateAssert` is a `C++`-style variadic function and supports the standard `printf` format specifiers. This method was found to be the most performant at JIT-compilation and when combined with **`AddressSanitizer`** the framework is capable of intelligently reporting the

cause of many critical memory errors at run-time. To do so, the `CBuilder` shadows every memory-touching operations (e.g., `CreateLoad`) supplied by the `IRBuilder` and automatically inserts the appropriate assertion calls.

Because of the nature of many of the kernel `DoSegment` generator functions, many assertions are redundant. Even though the goal of using asserts is not to generate code quickly but rather to diagnose a potential fault or logic error, Parabix includes a custom `RemoveRedundantAssertionsPass` LLVM module pass to eliminate or report statically proven checks (depending on their truth assignment) and remove assertions that are strictly dominated by one that tests the same condition, including ones in which two aliasing pointers check whether the same fixed-length region of memory is poisoned.

Unfortunately, while this method worked well initially, recent `C++` compilers enable *position-independent code* by default even with statically-linked executables. Call stacks were stored as global variables in the cached object files but the return addresses from run to run almost always differ. Consequently, `addr2line` cannot correctly translate the addresses into file names and line numbers in such programs. Fixing this flaw is an important avenue of future work.

IDISABuilder

The `CBuilder` is extended by the family of `IDISABuilder` subclasses. With the exception of the `KernelBuilder`, which we discuss shortly, each `IDISABuilder` implements an architecture-specific variation of the IDISA library. For each targeted architecture, IDISA provides a wide variety of SWAR operations for every power-of-2 field width less than or equal to the SIMD-register width (i.e., *block_width*). These operations are coarsely divided into vertical, horizontal, expansion, field movement and full register groups. We provide only a brief overview of the IDISA library here and refer the interested reader to [55] and [26].

- **Vertical operations** encompass most of the mathematical operations, such as `simd_add`, `simd_sub` and `simd_xor`. As shown in Figure 2.9, most vertical operations are lane-based binary intrinsics, performing some specific operation on each field width sized lane. Unary vertical operations, such as `simd_not`, take only one operand but function similarly.

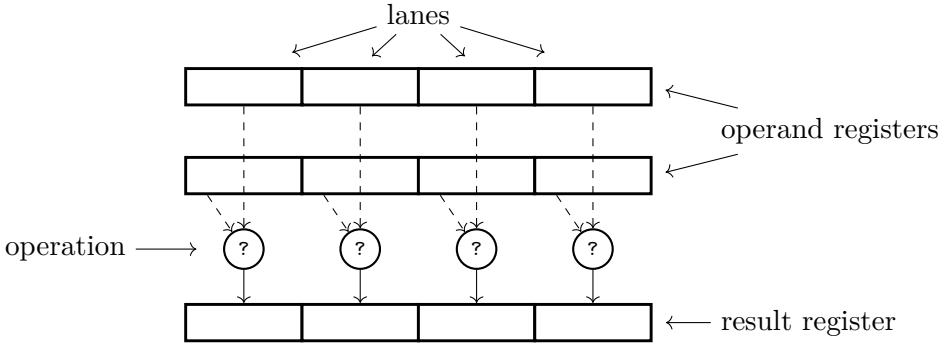


Figure 2.9: Vertical IDISA operations (Adapted from [55])

• **Horizontal operations** vary more in function but generally accept one or two operands and compress and/or concatenate them in some fashion. For example, the unary operation `hsimd_signmask` packs each of the most significant bits together into a standard register. The binary operation `hsmid_add_hl` however, adds pairs of adjacent lanes from two registers together to product a result, as shown in Figure 2.10.

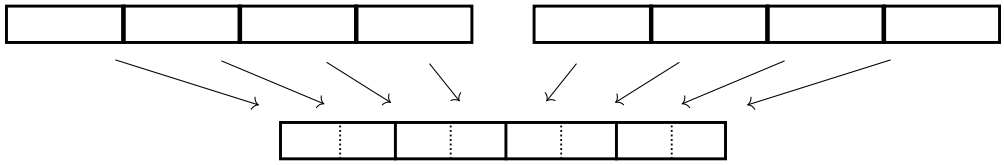
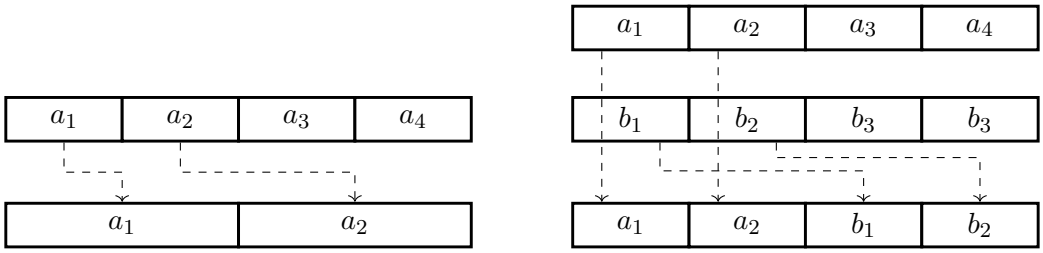


Figure 2.10: Horizontal IDISA operations (Adapted from [55])

• **Expansion operations** take one or two operands but every such operation uses only half the lanes from each operand to produce a result. For example, `esimd_zeroextendl` will zero extend the lower two lanes of Figure 2.11a, returning a value that doubles the field width of each lane. `esimd_mergel`, however, shown in Figure 2.11b, takes the low half of two operands and combines them in order in the result, preserving the width of each lane.



(a) Unary (b) Binary

Figure 2.11: Expansion IDISA operations (Adapted from [55])

• **Field movement operations** contain most of the horizontal inter-lane operations. For example, unary operations `mvmd_extract` and `mvmd_splat` will respectively write a single lane to a general register and broadcast a single register across all $(^{blockwidth\_width}/_{field\_width})$ lanes. As shown in Figure 2.12, `mvmd_dsrli` "concatenates" its operands and right shifts both of them by a fixed number of lanes before returning the lower half of the combined register.

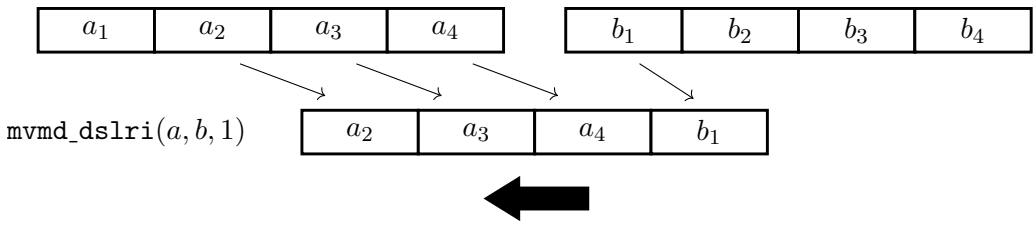


Figure 2.12: Field movement IDISA operations (Adapted from [55])

• **Full register operations** encompass all of the operations that strictly operate on the full SIMD register. The `PabloCompiler` (discussed in §2.3.4) relies on these functions to

generate performance critical code. Many of these operations are straightforward, such as `bitblock_load`, `bitblock_store` and the `bitblock_any` test, which checks whether any lane is non-zero. More complex operations, such as `bitblock_add_with_carry`, are often composed of other IDISA operations.

Nearly every IDISA operation can be represented with architecture-independent LLVM instructions but [77] noted such instructions could rarely be lowered to a native machine instruction and when lowering did not result in an exact mapping, LLVM's backend compiler almost always generated scalarized code (e.g., implementing `bitblock_add_with_carry` — one of the most important functions for `Pablo` — with two `@llvm.uadd.with.overflow` intrinsics produced code that was $\sim$ 20% slower in `AVX-2` machines than its "optimally-lowered" IDISA version.) To avoid customizing the LLVM instruction selection process, each `IDISABuilder` generates "pre-lowered" IR. The disadvantage of "pre-lowering", however, is it is both less precise than correctly selecting the best native instructions and increases both the LLVM optimization and compilation time due to an often substantial increase in code size. Improving this process and the "optimality" of the IDISA instructions (potentially via offline stochastic superoptimization [103]) are avenues of future research.

Many more variations of the IDISA library exists within the Parabix framework than what is depicted in Figure 2.8. The `JITDriver` (discussed in §2.1.11) automatically detects the current architecture and CPU features at start-up and chooses the best `IDISABuilder` subclass for the system. This permits programmers to write generic SIMD IR that will be JIT-compiled into the most efficient code supported by the CPU. Note, however, that `NVPTXBuilder` is not automatically instantiated by the `JITDriver`. The `NVPTXBuilder` is an experimental CUDA IR builder, is designed to generate PTX IR for Nvidia GPGPUs. Currently, the Parabix framework cannot determine which kernels might benefit from GPGPU acceleration. Providing full support for GPGPUs acceleration is an avenue for future work. For more information on the `NVPTXBuilder`, we direct the reader to [75].

KernelBuilder

The `KernelBuilder` provides an API for accessing scalar field values and streamset I/O data and properties, such as a processed, produced and an available item counts. All dataflow-related functionality is relegated to the `KernelBuilder` class.

To provide a single interface in the remainder of the framework, only the `JITDriver` is aware of which `IDISABuilder` was chosen when instantiating the `KernelBuilderImpl`. The `Impl` is always passed as an abstract `KernelBuilder` object. Virtual inheritance of the templated `IDISABuilder` type eliminates the diamond problem. This organization, while incapable of supporting devirtualization of IDISA methods, provides the greatest flexibility for future expansion (e.g., while not implemented for the current framework, IDISA intrinsics were developed for the ARM architecture [55] and could be added with little effort.)

2.1.9 Kernel compiler

Although every kernel, including the pipeline, is eventually compiled or otherwise mapped to machine code, every kernel must be compiled to LLVM IR at least once. Generating the code for each kernel method introduces a great deal of state to the `C++` kernel objects. For example, to allow the `KernelBuilder` to refer to scalars and streamsets by binding name, at least one map must be appropriately populated with each string and `llvm::Value` pointer pair. Specialized kernel compilers, such as for the pipeline, generate numerous analysis graphs once JIT-compilation begins.

The state required for JIT-compilation is all-but unnecessary to identify a cached instance of a kernel and following the first execution of a program, we expect almost every kernel will be loaded directly from the `.cache`. Since our objective is to provide the greatest run-time performance possible (without introducing undefined behaviour), we separate all JIT-compilation responsibilities from the kernel classes and instead have each kernel object instantiate their own kernel compiler as required. The kernel compiler then calls on the generator functions of the kernel object after generating any necessary boilerplate code.

2.1.10 Pipeline compiler

The `PipelineCompiler` is the focus of this dissertation and described in detail in Ch. 4. For this chapter, it suffices to view it is a specialized `KernelCompiler` whose responsibility is to transform the abstract kernel and streamset representation of a Parabix program (as provided by the programmer) into LLVM IR module that fits the standard `DoInit`, `DoSegment` and `DoFinalize` method structure.

2.1.11 JIT driver

Although the responsibilities of the `JITDriver` class (and its subclasses) are relatively small, apart from writing the overridable kernel generator methods, the `JITDriver` is the central point of interface to our framework. Through this class, at edit-time a programmer can construct kernel objects and define their streamset and scalar I/O relationships.

At start-up, the `JITDriver` also checks the hardware characteristics and determines the appropriate `KernelBuilder` implementation to provide to the programmer. Even if no LLVM IR is generated during JIT-compilation, the resolved `KernelBuilder` type-code is required to locate any precompiled instances. Once the `JITDriver`'s `compile` method is called, we begin the JIT-compilation subphase of the program. Based on the program signature, defined either by a custom pipeline name or by the kernel I/O and accompanying attributes, the `JITDriver` determines the signature for the pipeline object. Afterwards, it dispatches to the `ParabixObjectCache` to determine whether a precompiled version of the pipeline or any of the kernels already exists. Should any be present, the `JITDriver` fetches them from the `.cache`. Uncached kernels are gathered and independently optimized using

standard LLVM IR passes then JIT-compiled to machine code before being automatically inserted into the `.cache` (unless caching is disabled.) Currently JIT-compilation relies on `MCJIT` due to its relative stability between differing LLVM versions but this may be re-assessed in the future. JIT-compilation is also single threaded but incorporating separate compilation into the `JITDriver` is merely an implementation challenge. However, given the caching infrastructure, it is questionable whether it is necessary after the initial cold run of each Parabix application. Investigating this is an avenue of future work.

After compiling all of the kernels, the `JITDriver` inspects the pipeline and determines whether it contains any kernels with a `Family` attribute. If so, the `JITDriver` constructs and compiles a `main` function for the pipeline and returns it to the programmer. Otherwise the pipeline will contain a precompiled `main` function, which the `JITDriver` links to instead.

> **Note:** the `JITDriver` owns the `llvm::ExecutionEngine` that contains all of the JIT-ed code and must persist until the final completion of the JIT-execution subphase.

Currently, two subclasses of the `JITDriver` exist: the `CPUDriver` and the `NVPTXDriver`. The former is for commodity hardware and supports any that are natively handled by LLVM. The latter was intended to support the `NVPTXBuilder` to interface with the Nvidia CUDA compiler. For more information on the `NVPTXDriver`, we refer the reader to [75].

2.2 Common Parabix kernels

To facilitate the development of Parabix programs, a wide variety of kernels have already been included in the framework. The following is an inexhaustive category-grouped list of the major ones currently supported:

2.2.1 I/O kernels

The Parabix framework is intended to generate efficient text parsing and transcoding programs. To support this, the framework provides several file I/O source and sink kernels. The two main source kernels are the **`MMapSourceKernel`** and **`ReadSourceKernel`**. A third option, the **`FDSourceKernel`**, is a combination of both that dynamically dispatches to the best option depending on the input file stream type, discussed below.

As its name implies, `MMapSourceKernel` relies on the `mmap` capabilities of the underlying OS to enable memory-mapped file I/O. It marks its output streamset as a `ManagedBuffer` because the memory it returns is directly provided by the OS — with one significant caveat that we will address shortly. Even though the OS lazily releases the least-recently used pages of memory-mapped data on its own, the `MMapSourceKernel` directly informs the OS which portions are fully consumed (and therefore releasable) via `madvice` to reduce the allocated page space (opposed to its addressed space.) Because `mmap` maps files directly to user memory space, it avoids the inherent user/kernel mode switching associated with a `read`

system call, which is the backbone of the `ReadSourceKernel`. Regrettably, `mmap` only works with regular files, meaning that `stdin`, device, and virtual files, such as `/proc/cpuinfo`, cannot be memory-mapped. Such files must be processed using the `ReadSourceKernel`.

To replicate the linear access to data provided by `mmap`, the `ReadSourceKernel` utilizes an internal copyback mechanism to read in $4KB$ chunks per invocation and discard *block_width*-aligned bytes based on the consumed item count whenever the requested output exceeds its internal buffer. Despite the potential page faults incurred by `mmap`, our preliminary assessment of both kernels shows that `MMapSourceKernel` produces $\sim 2\times$ the bytes per second relative to the `ReadSourceKernel` when processing files of $\geq 1MB$ but has not been tested against smaller files in detail.

Safely using memory-mapped files in Parabix comes with a caveat: the OS maps data a page at a time but accessing data past the `EOF` risks incurring a bus error when said access exceeds the final page boundary. To ensure safe access to the data, the `MMapSourceKernel` allocates and returns an copy of the last page of data (plus any unconsumed data) when the last byte of data falls on a page-aligned boundary in *anticipation* that a subsequent portion of the program may read past the `EOF`. Although this sounds unusual, it is done to handle a common problem in Parabix programs. Many `PabloKernels` (discussed in depth in §2.3) use intrinsics that leave a cursor after the `EOF`. Such "supplementary" positions are explicitly reported to the pipeline via an `Add` attribute but that information is not passed to any subsequent kernel. Consequently, any kernel that correlates cursor positions with memory-mapped data risks reading past the `EOF` unless they actively check for this case, imposing a constant overhead on the program. This functionality is woven into the logic of the `ReadSourceKernel` and the as-of-yet unmentioned `MemorySourceKernel`.

The `MemorySourceKernel` forwards an input streamset as its output except when it reaches its last segment wherein for the same reason mentioned for the `MMapSourceKernel`, it makes a safe copy for the rest of the pipeline. It is primarily intended as an adapter for any potentially-unsafe input streamset to the pipeline and unlike the `MMapSourceKernel`, it always allocates $num\_of\_required\_bytes + 8 \times block\_width$ for its final segment.

Currently Parabix supports two types of output kernels: `StdOutKernel` and `FileSink`. Both of these kernels transfer data via a `write` system call. Using `mmap` for file output has not been explored yet. As the name implies, `StdOutKernel` writes directly to the `stdout` communication stream. `FileSink` instead writes to a user-defined file but initially generates a temporary file to prevent overwriting an existing file during an errant execution.

2.2.2 Pablo-support kernels

The `PabloKernel` itself is a significant enough component of this framework that we devote §2.3 to it. From the perspective of the pipeline and other kernels, a `PabloKernel` is one that operates primarily on streamsets of 1-bit wide stream items. We refer to such streams as *bit streams*. Although `Pablo` supports inputs of any uniform code-unit width, most `Pablo`

intrinsics operate strictly on bit streams. Since file I/O rarely comes in bit stream form, effectively using `Pablo` typically requires one or more of supporting kernels. Perhaps the most important of them are the two **transposition** kernels: `S2PKernel` and `P2SKernel`.

String	1	0	<	P	p	l	/	>
ASCII	00110001	00110000	00111100	01010000	01110000	01101100	00101111	00111110

b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8
0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0
1	1	1	0	1	1	1	1
1	1	1	1	1	0	0	1
0	0	1	0	0	1	1	1
0	0	1	0	0	1	1	1
0	0	0	0	0	0	1	1
1	0	0	0	0	0	1	0

Figure 2.13: Byte-space to Bit-space Transposition

In a typical Parabix program, each code point of a file I/O character sequence contains a fixed or variable number of code-units (e.g., a `UTF-8` code-unit is 8-bits and each code point currently requires between 1 and 4 of them.) The **`S2PKernel`** transforms n-bit wide serial data into n parallel bit streams, $\mathtt{b}_1, \mathtt{b}_2, \ldots, \mathtt{b}_n$, s.t. the $j^{\mathtt{th}}$ bit-value of each $\mathtt{b}_i$ is the $i^{\mathtt{th}}$-bit value of the $j^{\mathtt{th}}$ code-unit of the input stream. The **`P2SKernel`** reverses this transformation. For example, in Figure 2.13 we translate the ASCII input "`10<Ppl/>`" into its 8-bit stream transposed equivalent. Further details both transposition algorithms can be found in [23].

The next two support kernels are the **`ScanMatchKernel`** and **`MatchCoordinatesKernel`**. Although the `P2SKernel` can transform bit-space data back into code-unit space, this is typically only required for transcoders. For example, `icgrep` is a RE matching program that may report variable-length chunks of the original file input data whenever a matching line is found. To correlate bit markers with with code-unit positions and return the source text, the both kernels rely on the count leading zeroes (`ctlz`) instruction. `ctlz`, however, only operates on 64-bit fields thus has to be chained together to identify any positions. Once a position is found, it is cleared and `ctlz` may be called again to find the next position (if any exists.) Since `icgrep` expects few matches, it utilizes a two-level scanning technique that amortizes the cost of calling `ctlz` by first packing up to 64 non-zero tests into a 64-bit wide field to branchlessly determine whether any matches were found and locate the field in which any exist. The `ScanMatchKernel` is a `SideEffecting` sink that relies on having a `C++` callback function to provide the `C++` program with a pointer, length and line number for

each match. `MatchCoordinatesKernel`, however, produces an output streamset consisting of one 64-bit integer coordinate for every matched position.

2.2.3 Filtering kernels

Filtering implies the compaction or selective insertion of data values. Although many methods of filtering have been explored in Parabix, the most common are now the filter by mask pair, `FieldCompressKernel` and `StreamCompressKernel`, and those intended to operate within swizzled space, defined shortly.

Input stream:	`abcdefghjklmnpqrs`
Mask stream:	`...1.1...111..1..`
Output stream:	`dfklmq`

Figure 2.14: Filter by mask example

Filter by mask is a straightforward filtering approach. Shown in Figure 2.14, given an input and a mask streamset of 1-bit wide items, this pair of kernels produces an output streamset whose length is precisely the number of 1-bits in the mask stream. Consequently, its output production rate is the `PopCount` of its mask stream. Any input streamset comprised of multiple streams will have the same mask applied to each. This amortizes some of the work but the most expensive functions must still be applied to each stream.

The `FieldCompressKernel` relies on the `pext` instruction (or Warren's parallel-prefix compression algorithm [112]) to extract masked bits from the input and place them into the contiguous least-significant bit positions of its output. `pext`, however, only operates on fields of 32 or 64-bit wide fields. To convert the field-width grouped packed output of `FieldCompressKernel` to a contiguous sequence of bits, the `StreamCompressKernel` essentially constructs a FIFO queue of output bits but because of its branch-free nature (excluding the outer-most data iteration loop) it is somewhat more complex than its purpose implies. It divides its work into four stages:

1. First it performs a lane-based parallel `popcount` followed by a lane-aligned partial sum computation. The popcounts indicate how many contiguous bits are in the least-significant positions of each input lane and the partial sum stipulates where those bits will eventually be placed in the output queue.

2. Using the partial sum, the `StreamCompressKernel` computes a sequence of offsets indicating how much to shift the data within each lane. During this step, the data is not moved to its final lane. Instead the shifts occur within the current lane with the proviso that because the output offset plus the "`popcount`-ed" length may cross a field boundary, data that would be shifted past the end of any field is instead placed into the subsequent

lane (or the overflow for next loop iteration.) Note that bit collisions cannot occur during this step.

3. Based again on the partial sum, the final output lane positions of each input lane are determined. The input lanes are compacted into a contiguous sequence s.t. whenever two input lanes are destined for the same output lane, the values of the input lanes are combined into the earliest possible input lane.

4. Finally, the input lanes are shifted into their final lane positions, any pending data from the prior loop iteration is combined with the new output, and the pending data for the next iteration is generated. Because we are not guaranteed to completely fill the output element of any stream, this pending data may actually be a copy of the current output that will be effectively overwritten on the subsequent iteration.

Swizzling is a common technique in GPGPU computations, typically represented as some form of $n \times n$ matrix transformation that is applied to data to improve cache locality and potentially reduce inter-warp dependencies. In Parabix, **swizzling** is an invertible lane-based SIMD function that is applied to a streamset S of n streams whose items can be evenly divided into n 32 or 64-bit lanes. Currently, the only practical use of swizzling is to compute S^{T} for each element set [115]. Obviously, any swizzle function whose streamset has fewer than n streams can substitute in an all $\overline{0}$s stream for its "missing" items but the output must still be an n-stream streamset. Any streamset that is in (or expected to be in) swizzled form must be annotated with the `BlockSize` attribute to be correctly handled by the pipeline. Any matching streamset can be swizzled using the `SwizzleGenerator` but the `SwizzledDeleteByPEXTkernel` combines swizzling with filtering; its advantage is that it filters each input stream with the same mask stream, allowing it to replace the multi-step inter-lane shifting performed by the `StreamCompressKernel` with cheaper intra-lane shifting. An avenue of future work may be to generalize these concepts in a meta-kernel and programmatically detect when it is advantageous to use one form of filtering or the other.

2.2.4 Spreading kernels

In Parabix, spreading is the inverse of filtering. Whereas filtering tends to remove positions of an input stream to produce a shorter contiguous output, spreading transforms a contiguous input into a longer non-contiguous output. The combination of filtering and spreading kernels are fundamental components of Parabix-style transducers (e.g., `csv2json`.)

Like filtering, various spreading and depositing algorithms have been explored in Parabix. The one most commonly used is now the spread by mask pair of kernels. Given an input streamset and a mask stream, at each 1-bit indicator in the mask stream spread by mask will deposit the earliest undeposited input bit of the input stream. The input stream rate is a `PopCount` of the mask stream and the output stream is in one-to-one correspondence

with the `Fixed`-rate mask stream. Figure 2.15 shows an example that effectively reverses the filtering example of Figure 2.14. Note that the mask stream in both cases is identical.

Input stream:	`dfklmq`
Mask stream:	`...1.1...111..1..`
Output stream:	`...d.f...klm..q..`

Figure 2.15: Spread by mask example

Spread by mask is a sequence of two kernel calls: one to `StreamExpandKernel` followed by one to `FieldDepositKernel`. Suppose we can uniquely identify the position of a bit from the input stream in the output stream. The `StreamExpandKernel` moves input bits into the lane encompassing their eventual output position but permits input bits to placed into more than one adjacent output lane to simplify the movement logic. This is done with a combination of variable-length shifts and permutations and — like the `FieldCompressKernel` — is directed by a parallel `popcount` and partial sum calculation of the mask stream. `FieldDepositKernel` then takes this output and completes the process by depositing the "field-grouped" input bits into the correct output positions, confident in the knowledge that any extraneous (undeposited) bits will be correctly placed in a subsequent lane.

2.3 Overview of the Pablo architecture

One of — if not the — defining feature of the Parabix framework is the `PabloKernel`: almost all matching, parsing or transcoding actions within a Parabix program are heavily dependent upon one or more `PabloKernels` and the vast majority of all SIMD-parallelism within Parabix comes from exploiting `Pablo` strengths. Figure 2.16 provides a simplified and incomplete overview of the current `Pablo` architecture. Notably missing are the subclasses of the `PabloAST` class, which can be inferred by the instructions provided by the `Pablo` language, and the optimization passes, discussed in §2.3.4. Before describing the `PabloKernel` and its accompanying compiler, we first define the `Pablo` language.

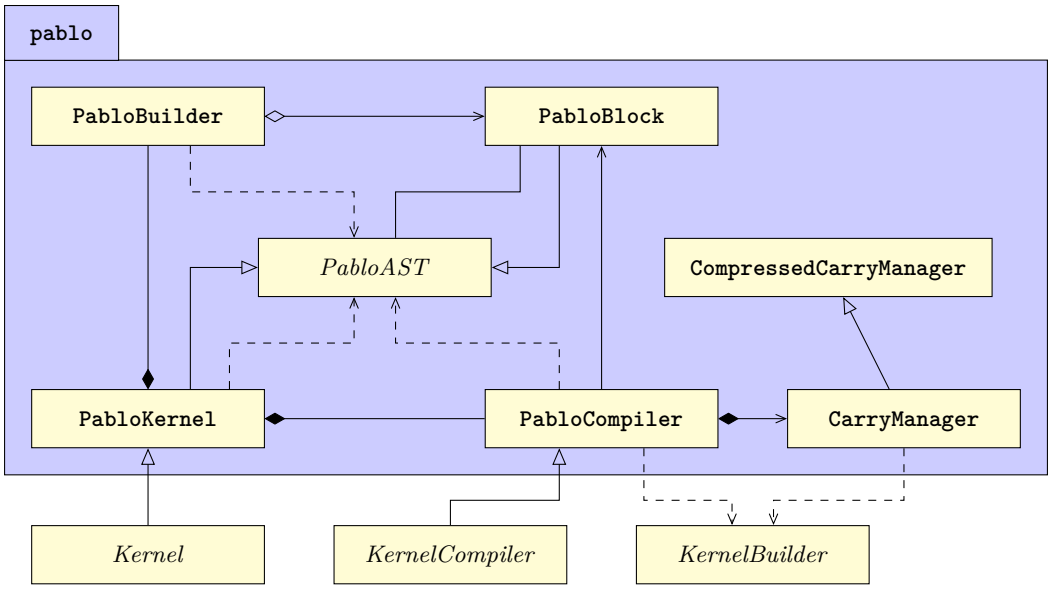


Figure 2.16: Simplified Pablo compiler UML class diagram

2.3.1 Pablo language

Although `Pablo` supports items of any uniform bit-width, the inputs to almost every `Pablo` instruction are either `Integer` scalars or bit streams. To the framework, a **bit stream** is a sequence of 1-bit fields stored in a single stream of a n-ary streamset. From `Pablo`'s perspective, however, a bit stream represents an unbounded sequence of bits that is simultaneously viewed as an arbitrary-large integer and a contiguous array of Boolean variables. From this point of view, every `Pablo` instruction is performed on the entire input concurrently. Moreover, the length of all bit streams within a `Pablo` program are identical, including those derived by any `Pablo` expression. In practice, when the `DoSegment` method of a `PabloKernel` is invoked, inputs are processed one *block_width* at a time. As we discuss later in §2.3.4, the `PabloCompiler` ensures the compiled LLVM IR faithfully mimics the unbounded `Pablo` code. For now — unless unavoidable — we ignore the complications imposed by commodity hardware and instead focus on unbounded model.

⟨BitStream⟩ ⊨ ⟨Var⟩ | ⟨Constant⟩ | ⟨Expression⟩

⟨Expression⟩ ⊨ ⟨Binary⟩ | ⟨Lookahead⟩ | ⟨UnaryWithInteger⟩
| ⟨IndexedAdvance⟩ | ⟨TernaryWithOpCode⟩
| `Not` ⟨BitStream⟩

⟨Constant⟩ ⊨ `Zeroes` | `Ones` | `Repeat` ⟨Integer⟩ ⟨Integer⟩

⟨Binary⟩ ⊨ ⟨BinaryFuncType⟩ ⟨BitStream⟩ `,` ⟨BitStream⟩

⟨BinaryFuncType⟩ ⊨ `And` | `Or` | `Xor` | `MatchStar` | `ScanThru` | `SpanUpTo`

⟨Lookahead⟩ ⊨ `Lookahead` ⟨Var⟩ `,` ⟨Integer⟩

⟨UnaryWithInteger⟩ ⊨ ⟨UnaryWithIntegerFuncType⟩ ⟨BitStream⟩ `,` ⟨Integer⟩

⟨UnaryWithIntegerFuncType⟩ ⊨ `Advance` | `EveryNth` | `Extract` | `TerminateAt`

⟨IndexedAdvance⟩ ⊨ `IndexedAdvance` ⟨BitStream⟩ `,` ⟨BitStream⟩ `,` ⟨Integer⟩

⟨TernaryWithOpCode⟩ ⊨ `Ternary` ⟨Integer⟩ `,` ⟨BitStream⟩ `,` ⟨BitStream⟩ `,` ⟨BitStream⟩

⟨Assign⟩ ⊨ `Assign` ⟨Var⟩ `,` ⟨BitStream⟩ | `Assign` ⟨Var⟩ `,` ⟨Var⟩

⟨If⟩ ⊨ `If` ⟨IfCondition⟩ `{` ⟨StatementList⟩ `}`

⟨IfCondition⟩ ⊨ ⟨BitStream⟩ | ⟨IntegerComparison⟩

⟨While⟩ ⊨ `While` ⟨WhileCondition⟩ `{` ⟨StatementList⟩ `}`

⟨WhileCondition⟩ ⊨ ⟨Var⟩ | ⟨IntegerComparison⟩

⟨StatementList⟩ ⊨ ⟨Statement⟩ ⟨StatementList⟩ | ϵ

⟨Statement⟩ ⊨ ⟨Assign⟩ | ⟨If⟩ | ⟨While⟩ | ⟨Expression⟩

Grammar 2.1: Pablo bit stream operation AST BNF grammar

Grammar 2.1 presents an overview of this section. Notably missing is definition of a `Var`, which is simply a named entity, and the intrinsics that do not operate on bit streams, which we append later in Grammar 2.5. **Note:** `Pablo` is a typed language and by design does not support either implicit or explicit type recasting.

Before describing the individual `Pablo` intrinsics, we first define how kernel I/O is represented in `Pablo`. Every streamset relationship associated with a `PabloKernel` is represented as a `Var` object. Such `Vars` can be referred to either by their port number or by name. Recall that streamsets can contain multiple streams. Every I/O derived `Var` is an array of streams but any `Pablo` intrinsic with a bit stream operand accepts only a singular bit stream value.

To refer to a specific stream, we use the `Extract` expression to create a reference to a singular element of a stream array. This `Extract`-ed value can be used as a concrete value for any intrinsic that accepts a stream of that element type or — in the case of an output `Var` — may be assigned a value with the `Assign` statement, which will write the particular value to the appropriate stream of the corresponding output streamset. Programmers are not limited to I/O `Vars`. They can be created at any time and used to represent potentially *mutable variables*. As such, they are a fundamental component of any `Pablo` program.

`Vars`, `Assign` and `Extract`

A `Var` is a potentially **mutable** object. Every `Var` that refers to an I/O port is marked as read/write only, respectively. Assuming the type of V and E match, **`Assign`(V, E)** conceptually writes the value E to the `Var` V. `Extract` AST nodes are an unnamed subclass of `Var`. Presuming the streamset referred to by `Var` object V contains k or more streams, **`Extract`(V, k)** returns a reference to the k^{th} stream V, where k is a constant `Integer`. `Extract` nodes inherit any I/O restrictions from their source `Var`.

Although conceptually simple, `Vars` are multipurpose. A programmer can create an internal `Var` objects at any time. Every internal and output `Var` is considered mutable since they can all be that can be assigned (or reassigned) a new value via the `Assign` statement. All `Vars` are function-scoped but internal and output `Var` objects are initially undefined until given a value with an `Assign`. Any operation that reads a `Var` will see its most recently `Assign`-ed value w.r.t. the order of operations executed by the processor at run-time. Reading an undefined `Var` is prohibited and thus `Vars` must be defined along every path that reads them. Partial support for internal `Var` arrays was explored but not fully generalized to support dynamically-resizable array constructs.

Although external I/O `Vars` directly reference their backing streamsets, internal `Vars` are typically placeholders for what will eventually become LLVM ϕ-nodes. A detailed discussion of ϕ-nodes is beyond the scope of this dissertation; we refer the interested reader to [30]. For the purpose of this section, ϕ-nodes can be viewed as mappings between `Vars` and their current concrete value. Any AST node that reads a `Var` will read its concrete value and any that writes to it, will write to directly to the backing streamset buffer or update its ϕ-node.

Even though `Pablo` is a pointer-free language, aliasing is technically possible since two I/O `Vars` could point to the same streamset and two `Extract` nodes could refer to the same stream. Currently, no consideration for aliasing is given by the `PabloCompiler`.

The next important topic to consider is the notion of control flow. `Pablo` is a simple language but does support a very restricted notion of branching. Specifically, `If` and `While` branch statements. Both of these are standard entry-controlled constructs whose condition guard can be either an integer scalar comparison or an implicit non-zero test for a bit stream. An important caveat here is that there is **no `Else` branch**. Consequently, `If`s are commonly treated as *optimization guards* rather than for traditional control flow.[6] `While` loops, however, follow their widely-accepted definition. With one proviso (which we discuss

[6]Given `Pablo` programs P and Q, where Q is generated from P by replacing the condition variable of an `If` statement S with `TRUE`, S is an *optimization guard* if and only if $P \neq Q \wedge \texttt{output}(P) \equiv \texttt{output}(Q)$ for every possible input to P/Q, where `output` returns the set of `Var` assignments.

after introducing the `MatchStar` intrinsic) any `Pablo` program whose sequential logic cannot be represented with an acyclic FSM requires a `While` loop.

`If` and `While` control flow branches

`Pablo` provides both **`If`** branch and **`While`** loop control flow constructs. Instead of having branch targets, both `If`s and `While`s contain a simple nested body. Each body is a statically-scoped `PabloBlock` but to increase their utility, any `Var` that is assigned a value prior to the `If` or `While` statement and within the nested body is said to escape the nested block. Any operation that reads a `Var` will see its most recently `Assign`-ed value w.r.t. the order of operations executed by the processor at run-time but reading an undefined `Var` is undefined behaviour. **Note:** to avoid an eventual infinite loop, it is the programmer's responsibility to ensure a `While` condition is an escaped `Var`.

Now that we have described the basic structure of a `Pablo` program, it is time to discuss the individual intrinsics. As mentioned in §2.2.2, utilizing `Pablo` typically requires us to first transpose the source text into n bit streams, where n is the code-unit bit-width of the raw data. Immediately following transposition is often a character-classification stage. Utilizing only SIMD bit-wise operations, **`And`** ($\wedge$), **`Or`** ($\vee$), **`Not`** ($\neg$) and **`Xor`** ($\oplus$), it is possible to identify a wide variety of characters in parallel. For example, an `ASCII` character is a left-angle bracket `'<'` if and only if $\neg(b_0 \vee b_1) \wedge (b_2 \wedge b_3) \wedge (b_4 \wedge b_5) \wedge \neg(b_6 \vee b_7) = 1$. Similarly, a character is numeric `[0-9]` if and only if $\neg(b_0 \vee b_1) \wedge (b_2 \wedge b_3) \wedge \neg(b_4 \wedge (b_5 \vee b_6)) = 1$. Note the potential code reuse for $\neg(b_0 \vee b_1) \wedge (b_2 \wedge b_3)$. Apart from the bit-wise operations, `Pablo` also supports various **`Ternary`** instructions, which are essentially pair-wise combinations of the above binary expressions that take three operands. These instructions have native support starting with `AVX-512` but can be encoded with the aforementioned operations [111].

Boolean logic operations

The binary `And`, `Or` and `Xor` functions and unary `Not` function all follow their standard Boolean algebra definitions and `Sel`(C, T, F) is what is commonly referred to as the if-then-else operator: $(C \wedge T) \vee (\neg C \wedge F)$. `Ternary` operations take an 8-bit `Integer` code c and three bit stream operands, labelled X, Y and Z. Each c_i variable in Tbl. 2.1 corresponds to result returned by the `Ternary` for the combination of aligned bit-values on the i^{th} row. The code-value c is computed from the c_i variables as follows:

$$c = \sum_{i=1}^{8} c_i \times 2^{(i-1)}$$

X	Y	Z	
0	0	0	c_1
0	0	1	c_2
0	1	0	c_3
0	1	1	c_4
1	0	0	c_5
1	0	1	c_6
1	1	0	c_7
1	1	1	c_8

Table 2.1

e.g., `Ternary`$(\texttt{0x80}, X, Y, Z) \equiv X \wedge Y \wedge Z$ and `Ternary`$(\texttt{0x96}, X, Y, Z) \equiv X \oplus Y \oplus Z$.

Parsing with Pablo requires a richer set of language features. While Boolean logic alone can classify ASCII characters, even lexing UTF-8 characters (a variable-length encoding format) requires stateful transitions between codeunits. In Figure 2.17, we consider a very simple case: find matching instances of "<[a-zA-Z]+>". L_0 to L_2 are the obvious character classifications but to calculate R, we need to introduce some unique arithmetic-based intrinsics: Advance and ScanThru. Note that **for visual simplicity we assume a left-to-right big-endian ordering** in Figure 2.17 and the other figures in this subsection even though our targeted platforms are typically little-endian (i.e., we treat right-most as a synonym for most-significant and left-most with least.) Finally, bit stream variables with a 0-value are represented with a period '.' to highlight the significant lexical and syntactic positions.

	<a><VAlid> <StrIng> <>Ignored><Error]
L_0 = [<]	1..1.......1.........1.........1.......
L_1 = [a-zA-Z]	.1..11111...111111.....1111111..11111..
L_2 = [>]	..1......1........1...1.......1........
S_0 = Advance$(L_0, 1)$	.1..1.......1.........1.........1......
S_1 = $S_0 \wedge L_1$	.1..1.......1...................1......
S_2 = ScanThru(S_1, L_1)	..1......1........1..................1.
R = $S_2 \wedge L_2$	..1......1........1....................

Figure 2.17: Pablo program for <[a-zA-Z]+>

Advance

Advance(S, k) shifts each 1-bit of bit stream S forwards by k positions, where k is a static PabloAST integer. It is the simplest yet the most prevalent of the Pablo intrinsics. Conceptually, the result of this can be considered the "sum" of k additions of S.

ScanThru

Given a marker stream M and a cursor stream C, ScanThru(M, C) deposits any 1-bit of M at the first position to the right of each run of 1-bits in C.

M	..........1......1....1.....1.........
C	..111....11111....1....111...111111....
$T_0 = M + C$	..111....11...1....1......1........1...
$R = T_0 \wedge \neg C$	..............1....1......1........1...

Figure 2.18: $R =$ ScanThru(M, C) (Adapted from [24])

Regrettably, `Advance` and `ScanThru` alone cannot recognize all regular languages. Suppose we want to locate all tokens that match "`[ab]*bbb[ab]*`". This RE is inherently ambiguous because the `[ab]*` prefix and suffix could potentially overlap the three-`b` infix. Both `Advance` and `ScanThru` are insufficient since the prefix is unbounded and `ScanThru` would greedily consume all '`b`' characters. To handle such ambiguities, we introduce a new intrinsic, `MatchStar`, and depict a possible solution in Figure 2.19.

	aabbabaaaaab bbb babbabbbabbba baaabbbb
L_0 = `[a]`	.11..1.11111.......1..1...1...1..111.....
L_1 = `[b]`	...11.1.....1.111..1.11.111.111..1...1111.
L_2 = `[ab]`	.111111111111.111..1111111111111.11111111.
L_2 = ⟨whitespace⟩	1...........1...11...........1.......1
S_0 = `Advance`$(L_2, 1)$	.1...........1...11...........1........
S_1 = $S_0 \wedge L_2$	.1...........1....1...........1........
S_2 = `MatchStar`(S_1, L_2)	.1111111111111111.11111111111111111111
S_3 = $S_2 \wedge L_1$	...11.1.....1.111..1.11.111.111..1...1111.
S_4 = `Advance`$(S_3, 1)$	....11.1.....1.111..1.11.111.111..1...1111
S_5 = $S_4 \wedge L_1$	....1.........11.....1..11..11.......111.
S_6 = `Advance`$(S_5, 1)$	.....1.........11.....1..11..11.......111
S_7 = $S_6 \wedge L_1$	................1.........1...1........11.
S_8 = `MatchStar`(S_7, L_2)	................11........1111111......111
S_9 = $S_8 \wedge L_2$	................1.............1........1

Figure 2.19: Pablo program for `[ab]*bbb[ab]*` using MatchStar

`MatchStar`

Given a marker stream M and a cursor stream C, `MatchStar`(M, C) marks every position between the left-most 1-bit of M within the run of 1-bits in C up to and including the position immediately to the right of each run in C.

M	.1..........1...1........1.....
C	.111....1.......11111.....11.1..
$T_0 = M \wedge C$	.1.............1.........1.....
$T_1 = T_0 + C$	....1...1............1......11..
$T_2 = T_1 \oplus C$	.1111...........111111....111...
$R = T_2 \vee M$	.1111........1...111111....111...

Figure 2.20: $R =$ `MatchStar`(M, C) (Reproduced from [27])

There are two notable observations that should be considered when viewing Figure 2.19. Firstly, an obvious alternative exists that first identifies the infix and then determines whether it is preceded by the prefix using `ScanThru` is technically possible in the unbounded model but could require unbounded memory at run-time. As such, the `PabloCompiler` will not generate the correct code for that program since we define the need for unbounded memory is undefined behaviour even in the unbounded model. Secondly, a careful eye will note that each `MatchStar` in Figure 2.19 replaces what could have been implemented with `Advances` and `ScanThrus` inside a `While` loop. Although very simple `While` loops can be replaced with a `MatchStar`, the cases in which we can reduce a loop to a `MatchStar` are highly restrictive: at minimum, we must have a bit stream condition whose initial value is a superset of its subsequent (iterated) values and a loop-invariant contiguous cursor stream.

With these three operations, several subtle tricks are possible that highlight the importance of considering problems in both a "positive" and "negated" space. The first trick is shown in Figure 2.21. Suppose we wish to ignore any whitespace preceding the first character in an input. By far the simplest way to handle this in `Pablo` is to ensure there is a bit stream with a single 1-bit at its very first position then use `ScanThru` to the first character of interest. By `Advance`-ing a constant all ones stream by k positions and negating the result, this will leave a stream with precisely k 1-bits at the start of the bit stream. Although this idiom is not currently specialized by the compiler, it is common enough that it would be beneficial to explore ways to make this a "zero-cost" feature of the language. For simplicity, in the later examples we will assume that some significant marker will trigger the parsing process of every token. However, this technique can often be used to ensure a cursor exists at the start of the first token.

	ignored	not-ignored
Z = `Zeros`	.....	
$S_0 = \neg Z$	1111111111111111111111111111	
S_1 = `Advance`$(S_0, 1)$	.1111111111111111111111111111	
$S_2 = \neg S_1$	1.....	
L_0 = ⟨whitespace⟩	111111111.....111.....1	
S_3 = `ScanThru`(S_2, L_0)	.....1.....	

Figure 2.21: Pablo program to find the first non-whitespace character

The second trick is associated with the *bounded repetition problem*. Suppose we want to find all matches to "`C{2,8}`", where `C` is a character set of a fixed-length encoding. The naive way would be to chain 8 `Advances` together and select the ones whose lengths are appropriate. By treating this as a *mask elimination* problem, we can perform it with only 5.

Specifically, for any repetition of `C{m,n}`, we require exactly $\lceil \log_2 m \rceil + \lceil \log_2 (n-m) \rceil + 1$ `Advance` expressions. To do so, we divide the problem into two steps: the first on Ln. 2 of

Alg. 2 locates the required "`C{m}`" prefix in which Ln. 3 determines the last character of each. The second phase begins on Ln. 4 where the algorithm identifies any positions that fall after the "`C{n - m}`" suffix. Ln. 5 then determines whether the the distance between the required character and the next non-matching character (w.r.t. C) is less than $(n-m+1)$. An example of this can be seen in Figure 2.22.

Algorithm 2 Find consecutive runs of n to m non-whitespace characters in C

1: **procedure** ConsecutiveRange(C, m, n)
Require: $1 \leq n < m$
2: $S_1 \leftarrow$ IdentifyRun(C, m)
3: $S_3 \leftarrow$ Advance($\neg C, m$) $\wedge\ S_1$
4: $S_{10} \leftarrow$ IdentifyRun($\neg S_3, (n-m)+2$)
5: **return** MatchStar(S_3, C) $\wedge\ \neg(S_{10} \vee C)$
6: **procedure** IdentifyRun(C, k)
7: **if** $k = 1$ **then**
8: **return** C
9: **else**
10: $M =$ IdentifyRun($C, \lceil k/2 \rceil$)
11: **return** $M \wedge$ Advance($M, k - \lceil k/2 \rceil$)

		a ab abc abcde abcdefgh abcdefghi
C	= ⟨non-whitespace⟩	.1.11..111...11111.11111111.111111111.
S_0	= Advance($C, 1$)	..1.11..111...11111.11111111.111111111
S_1	= $C \wedge S_0$	....1...11....1111..1111111..11111111.
S_2	= Advance($\neg C, 2$)	..1.1..11...111.....1........1........
S_3	= $S_2 \wedge S_1$	....1...1.....1.....1........1........
S_4	= $\neg S_3$	1111.111.11111.11111.11111111.11111111
S_5	= Advance($S_4, 1$)	.1111.111.11111.11111.11111111.1111111
S_6	= $S_4 \wedge S_5$	.111..11..1111..1111..1111111..1111111
S_7	= Advance($S_6, 2$)	...111..11..1111..1111..1111111..11111
S_8	= $S_6 \wedge S_7$	...1........11....11....11111....11111
S_9	= Advance($S_8, 4$)	.......1........11....11....11111....1
S_{10}	= $S_8 \wedge S_9$	............................1........1
S_{11}	= MatchStar(S_3, C)	....11..111...11111.11111111.111111111
S_{12}	= $S_{10} \vee C$	.1.11..111...11111.11111111.1111111111
S_{13}	= $\neg S_{12}$	1.1..11...111.....1........1..........
S_{14}	= $S_{11} \wedge S_{13}$	.....1....1.......1........1..........

Figure 2.22: Pablo program to find tokens of $2-8$ non-whitespace characters

Not all `Pablo` instructions have a concise mathematical definition but they are none-the-less vital for efficient parsing of input data. These include `EveryNth` and `IndexedAdvance`. Let first us consider a simplified JSON-string grammar of Grammar 2.2 wherein we want to identify the starting and ending double-quote '`"`' positions bookending a ⟨value⟩ but ignore any escaped characters. Identifying a pair of quotes is difficult because we cannot distinguish between them at a lexical level but we can solve this problem easily with `EveryNth`.

$$
\begin{aligned}
\langle \text{char} \rangle &\models \texttt{Any codepoint except for \ or "} \\
\langle \text{escaped char} \rangle &\models \langle \text{char} \rangle \mid \texttt{\"} \\
\langle \text{value} \rangle &\models \langle \text{escaped char} \rangle \, \langle \text{value} \rangle \mid \epsilon \\
\langle \text{string} \rangle &\models \texttt{"} \, \langle \text{value} \rangle \, \texttt{"}
\end{aligned}
$$

Grammar 2.2: Simplified JSON string BNF grammar

		`"a1" \"a2 "paper\"s tricky\" example"`
L_0	= `["]`	`1..1..1...1......1.........1........1`
L_1	= `[\]`	`.....1..........1.........1..........`
S_0	= `Advance`$(L_1, 1)$	`......1..........1.........1.........`
S_1	= $L_0 \wedge \neg S_0$	`1..1......1........................1`
R_{end}	= `EveryNth`$(S_1, 2)$	`...1.............................1`
R_{start}	= $S_1 \wedge \neg R_{\text{end}}$	`1.........1........................`

Figure 2.23: Non-validating Pablo program for Grammar 2.2

`EveryNth`

The `EveryNth`(S, k) intrinsic takes a bit stream S and generates a result that marks every $k^{\texttt{th}}$ position with a 1-bit in S with 1 and leaves all others with 0. E.g. assuming $k = 2$:

S	`.1...........1...1.........1.....`
R	`.............1.............1.....`

Figure 2.24: $R =$ `EveryNth`$(S, 2)$

`IndexedAdvance` is related to `EveryNth` in that they both deposit markers at specific locations but is slightly more nuanced in its use. Suppose in Figure 2.25, ⟨index⟩ is a complex RE that matches any sequence of one or more (diacritic-agnostic) identical non-whitespace characters and results in a `Pablo` expression that leaves a 1-bit on the final character of each run. If we wish to locate any token composed strictly of $2 - 8$ ⟨index⟩ repetitions, we could replicate the matching function for ⟨index⟩ 8 times and store the appropriate results.

Since ⟨index⟩ is expensive, this is likely to be inefficient but by treating this as a variation of the bounded repetition problem, shown in Alg. 3, we require only 4 `IndexedAdvances`.

Algorithm 3 Find consecutive runs of n to m indexed positions

1: **procedure** ConsecutiveIndexedRange(I, W, n, m)
Require: $1 \leq n < m$
2: $S_1 \leftarrow$ IdentifyIndexedRun($I, I \vee W, m$)
3: $S_9 \leftarrow$ IdentifyIndexedRun($\neg S_3, (n-m)+2$)
4: **return** MatchStar(S_2, C) $\wedge \neg(S_9 \vee W) \wedge W$
5: **procedure** IdentifyIndexedRun(S, I, k)
6: **if** $k = 1$ **then**
7: **return** S
8: **else**
9: $M =$ IdentifyIndexedRun($S, I, \lceil k/2 \rceil$)
10: **return** $M \wedge$ IndexedAdvance($M, I, k - \lceil k/2 \rceil$)

	`aaaa ab abbc aabcde abcdddefgh abcdefghi`
L_0 = ⟨index⟩	`....1.11.1.11..11111.111..11111.111111111.`
L_1 = ⟨whitespace⟩	`1....1..1....1......1.........1.........1`
$S_0 = L_0 \vee L_1$	`1...111111.111.111111111..1111111111111111`
S_1 = `IndexedAdvance`$(L_0, S_0, 1)$	`.....1.11..111..11111.11..111111.111111111`
$S_2 = L_0 \wedge S_1$	`.......1...11...1111..11..11111..11111111.`
S_3 = `IndexedAdvance`$(S_2, S_0, 1)$	`........1...11...1111..1..111111..11111111`
$S_4 = S_2 \wedge S_3$	`............1....111...1..11111...1111111.`
S_5 = `IndexedAdvance`$(S_4, S_0, 2)$	`..............1...111.....111111...111111`
$S_6 = S_4 \wedge S_5$	`..................1.......1111.....11111.`
S_7 = `IndexedAdvance`$(S_6, S_0, 4)$	`....................1.......1111.....11`
$S_8 = S_6 \wedge S_7$	`.....................................1.`
$S_9 = L_1 \vee S_8$	`1....1..1....1......1.........1........11`
$S_{10} = \neg S_9$	`.1111.11.1111.111111.1111111111.11111111..`
S_{11} = `MatchStar`(S_2, S_{10})	`.......11..111..11111.1111111111.11111111.`
$S_{12} = S_{11} \wedge L_1$	`........1....1......1.........1.........`

Figure 2.25: Pablo program for ⟨index⟩$\{2, 8\}$ using `IndexedAdvance`

`IndexedAdvance`

An `IndexedAdvance`(S, I, k) is similar to a standard `Advance` in that it moves significant bits ahead by k positions. Each significant bit of the S stream is marked by a 1-bit in the index stream I and will be deposited k index positions ahead of their initial position. E.g.:

S `.a..b...c....d.e.f.g.......hi....`
I `.1..1...1....1.1.1.1.......11....`
R `........a....b.c.d.e.......fg....`

Figure 2.26: $R =$ `IndexedAdvance`$(S, I, 2)$

Some interesting parsing problems require the use of non-trivial constants to efficiently solve. Consider the full version of the JSON string recognition problem from [68], detailed by Grammar 2.3. In Figure 2.27, Langdale and Lemire generate a repeating even (E) and odd (O) bit sequence to identify the position following every odd-length run of `'\'`s in S_3 by independently detecting every `'\'` starting on an even position and ending on and odd or v.v. in S_1 and S_2 respectively.

⟨char⟩ ⊨ `Any codepoint except for \ or "`
⟨escaped char⟩ ⊨ ⟨char⟩ | `\\` | `\"` | `\n` | ...
⟨value⟩ ⊨ ⟨escaped char⟩ ⟨value⟩ | ϵ
⟨string⟩ ⊨ `"` ⟨value⟩ `"`

Grammar 2.3: JSON string BNF grammar

		"\\\n" """\\\\" "\"t\"" "false" "\\\\"
L_0	= ["]	1....1.111....1.1.1..11.1.....1.1....1.
L_1	= [\]	.111......1111...1..1...........1111..
E	= Repeat(0b10) → i2	.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.
O	= Repeat(0b01) → i2	1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1.1
S_0	= $L_1 \wedge \neg$Advance$(L_1, 1)$	.1........1......1..1...........1.....
S_1	= ScanThru$(S_0 \wedge E, L_1) \wedge O$	....1.............1....................
S_2	= ScanThru$(S_0 \wedge O, L_1) \wedge E$	......................1..............
S_3	= $S_1 \vee S_2$	....1.............1..1..............
S_4	= $L_0 \wedge \neg S_3$	1....1.111....1.1.....1.1.....1.1....1.
R_{end}	= EveryNth$(S_4, 2)$	.....1..1.....1.......1.......1......1.
R_{start}	= $S_4 \wedge \neg R_{\text{end}}$	1......1.1......1.......1.......1......

Figure 2.27: Pablo program for Grammar 2.3 (Adapted from [68])

Constants

In Pablo, there are two types of constants: those for non-negative Integer scalars and those for bit streams. Apart from having a fixed LLVM type of some $n = 2^m$ bit-width (denoted with in) Integer scalars have the standard definition. Every bit stream constant is an infinite repeating pattern. Such patterns can be created by the **Repeat(k)**, where k is a typed Integer. Due to their mathematical properties — especially during optimization phases — Pablo specializes the Repeat(0) → i1 and Repeat(1) → i1 sequences as Zeroes and Ones nullary expressions, respectively.

With problems like those depicted in Figure 2.23 – 2.27, we often want to generate a mask marking the positions between the start and end markers to filter out extraneous cursors from some bit stream. Although we can sometimes use MatchStar for such problems, when we can guarantee that for every start position there is exactly one end position and v.v. we have a far simpler arithmetic-based intrinsic SpanUpTo along with two obvious "syntactic sugar" extensions, InclusiveSpan and ExclusiveSpan. As shown in S_7 of Figure 2.28, this one-to-one restriction can be of an advantage sometimes for error detection. When multiple start positions precede an end, a "gap" will be left at each start position after the first. By leveraging this, we compute an error bit stream E_1 in Figure 2.28 in which the the first isolated 1-bit indicates an erroneous '<' in the element name and the subsequent 1-bits, an illegal tag end according to Grammar 2.4.

$$
\begin{aligned}
\langle \text{name char} \rangle &\models \texttt{Any codepoint except for < or >} \\
\langle \text{name} \rangle &\models \langle \text{name char} \rangle\ \langle \text{name remaining} \rangle \\
\langle \text{name remaining} \rangle &\models \langle \text{name char} \rangle\ \langle \text{name remaining} \rangle \mid \epsilon \\
\langle \text{value char} \rangle &\models \texttt{Any codepoint except for "} \\
\langle \text{value} \rangle &\models \langle \text{value char} \rangle\ \langle \text{value} \rangle \mid \epsilon \\
\langle \text{attrs} \rangle &\models \langle \text{whitespace} \rangle\ \langle \text{name} \rangle\ \texttt{="}\ \langle \text{value} \rangle\ \texttt{"}\ \langle \text{attrs} \rangle \mid \epsilon \\
\langle \text{tag} \rangle &\models \texttt{<}\ \langle \text{name} \rangle\ \langle \text{attrs} \rangle\ \texttt{>} \\
\langle \text{xml} \rangle &\models \langle \text{tag} \rangle\ \langle \text{optional whitespace} \rangle\ \langle \text{xml} \rangle \mid \epsilon
\end{aligned}
$$

Grammar 2.4: Simplified XML tag BNF grammar

	`<e a="" b="c">  <err<r a="<e a="><e> a=">"" >`
L_0 = `[<]`	`.1.............1...1.....1......1..........`
L_1 = `[>]`	`.............1.................1..1....1...1.`
L_2 = `["]`	`......11...1.1...........1.....1.......1.11...`
L_3 = `[=]`	`.....1....1.............1.....1.......1.......`
L_4 = ⟨whitespace⟩	`1..1....1......11......1.....1.......1......1.1`
S_0 = `EveryNth`$(L_2, 2)$	`.......1.....1.................1.........1....`
$S_1 = \neg S_0$	`1111111.11111.1111111111111111.111111111.1111`
$S_2 = S_1 \wedge L_2$	`......1....1.............1.............1..1...`
S_3 = `SpanUpTo`(S_2, S_0)	`......1....11............111111........11.1111`
$S_4 = \neg S_3$	`111111.1111..111111111111......11111111..1....`
S_5 = `Advance`$(L_3, 1)$	`......1....1.............1.....1.......1......`
$S_6 = \neg S_5$	`111111.1111.1111111111111.11111.1111111.111111`
$E_0 = S_2 \wedge S_6$	`.........................................1...`
$S_7 = L_0 \wedge S_4$	`.1.............1...1...........1..........`
$S_8 = L_1 \wedge S_4$	`.............1.................1..1.........`
S_9 = `SpanUpTo`(S_7, S_8)	`.1111111111111...1111.11111111111.11..........`
$S_{10} = S_9 \vee L_4$	`11111111111111.111111.11111111111.11.1......1.1`
$S_{11} = S_{10} \vee S_8$	`1111111111111111111.111111111111111......1.1`
$E_1 = \neg S_{11}$	`....................1.............111111.1.`
$E_2 = E0 \vee E_1$	`....................1.............111111.1.`

Figure 2.28: Pablo program for validation of Grammar 2.4 using `SpanUpTo`

SpanUpTo

`SpanUpTo`(S, E) computes a bit stream composed of regions demarcated by 1-bits in E. Returns a run of 1-bits starting from the left-most 1-bit in S to the end of the region but skips over the subsequent 1-bits in S or E. E.g.:

S	`1.........1..........1.............`
E	`...1.............................1`
$R = E - S$	`111.......111111111111.1111111111111.`

Figure 2.29: $R =$ `SpanUpTo`(S, E)

Not all problems in Parabix are purely matching problems. For example, to correctly report a matching line in `icgrep`, we must identify the *first* position of each linebreak then provide a "normalized" `EOL` character. In `ASCII`, a linebreak can be a `CR`, `LF`, or `CRLF` sequence. To determine whether a `LF` is part of a `CRLF` when parsing the `CR`, we require a `Lookahead`.

Lookahead

`Lookahead`(S, k) is the opposite of `Advance`(S, k) in that it shifts each 1-bit k positions *backwards*, where k is a static integer. Internally, however, its implementation differs considerably from an `Advance` because whereas `Advances` always propagates bit values forwards w.r.t. the ordering of the input sequence, a `Lookahead` must pull pre-computed data back. Consequently, S must refer to an input streamset variable of the `PabloKernel` whose input port has a `LookAhead` attribute of at least k positions. Because caching presumes that we bypass `Pablo` IR generation, it is the programmer's task to correctly attribute each port.

The power of `Lookahead` should not be undervalued. For example, Ken Herdy explored the concept of length-sorted symbol hashing [52], a technique that relied on `Lookahead`. With it, the goal is to group of similar-length tokens into an independent hash table in which each table has its own a unique hashing function. To perform length-sorted hash look ups, it is necessary to identify the start S_i and end E_i positions of each token within each $i^{\texttt{th}}$ group in the input stream. In Alg. 4, we present a modified version the *logarithmic-length partitioning* algorithm [52]. In it, each token of length k belongs to the $\lceil \log_2(k)+1 \rceil^{\texttt{th}}$ group. Alg. 4 assumes we can identify a superset of all start and end positions of each token as S and E, respectively. Since the Parabix framework executes kernels according to a fixed sequence (determined by the pipeline) we bound the problem to $n-1$ length-sorted groups, leaving the $n^{\texttt{th}}$ group to catch any remaining tokens.

Algorithm 4 Logarithmic length-sorting algorithm

1: **procedure** LOGLENGTHSORTING(S, E, n)
2: $R_0 \leftarrow S$
3: $M_0 \leftarrow E$
4: **for** $k \leftarrow 1$ to $n-1$ **do**
5: $L_k = \text{LOOKAHEAD}(M_{k-1}, 2^{k-1})$
6: $S_k = L_k \wedge R_{k-1}$
7: $E_k = \text{SCANTHRU}(S_k, \neg E)$
8: $R_k = R_{k-1} \oplus S_i$
9: $M_k = \text{ADVANCE}(M_{k-1}, 2^{k-1}) \vee M_{k-1}$
10: $S_n = R_{n-1}$
11: $E_n = \text{SCANTHRU}(S_n, \neg E)$
12: **return** $\{(S_1, E_1), (S_2, E_2), \ldots, (S_n, E_n)\}$

	Vel at tortor a inclusio
$R_0 = S$	1...1..1.....1.1.....
$M_0 = E$	...1..1.....1.1.....1
$L_1 = \texttt{Lookahead}(M_0, 1)$	..1..1.....1.1.....1.
$S_1 = L_1 \wedge \neg R_0$	.............1.....
$E_1 = \texttt{ScanThru}(S_1, \neg E)$	.............1.....
$R_1 = R_0 \oplus S_1$	1...1..1.....1.....
$M_1 = \texttt{Advance}(M_0, 1) \vee M_0$	...11.11.....1111.....1
$L_2 = \texttt{Lookahead}(M_1, 2)$	.11.11.....1111.....1..
$S_2 = L_2 \wedge \neg R_1$	....1..................
$E_2 = \texttt{ScanThru}(S_2, \neg E)$	.....1.................
$R_2 = R_1 \oplus S_2$	1.....1.....1.....
$M_2 = \texttt{Advance}(M_1, 2) \vee M_1$	...1111111...111111.....1
$L_3 = \texttt{Lookahead}(M_2, 4)$	111111...111111.....1....
$S_3 = L_3 \wedge \neg R_2$	1......................
$E_3 = \texttt{ScanThru}(S_3, \neg E)$	...1...................
$R_3 = R_2 \oplus S_3$	.....1.....1.....
$M_3 = \texttt{Advance}(M_2, 4) \vee M_2$	...1111111111111111111.1
$S_4 = R_3$	.....1.....1.....
$E_4 = \texttt{ScanThru}(S_3, \neg E)$	.............1.....1

Figure 2.30: Pablo logarithmic length-sorted partitioning algorithm example with $n = 4$

As discussed in §2.1.6, chaining kernels with `LookAhead` attributes comes at a cost. Each `Lookahead` intrinsic must refer to an input streamset `Var`. Consequently, every loop iteration of Alg. 4 will generate a separate kernel. Currently, the `PabloCompiler` can only generate a single `PabloKernel` at a time and does not have any facilities to automatically cut a `Pablo` program into multiple kernels. It is unknown whether it would be beneficial to automate this to simplify the construction of `Pablo` programs with `Lookahead`(s) or whether the complexity of supporting this in general is too great to match handcrafted code in performance. One strategy to mitigate this that has not yet been explored is to construct a prelude and postlude phases, both for the pipeline and kernels. With that and the ability for some instructions to effectively read a *block_width* ahead, it could be possible to support `Lookahead` operations on non-input `Vars` within a `PabloKernel`. It is an open question whether overall performance would benefit given these natural division points enforce finer-grained kernels for pipeline-parallelism (§2.1.5) to exploit.

Up to now, we have described intrinsics as they exist within the unbounded model. However, `Pablo` programs parse finite-length inputs. To deduce whether we have reached the end of the input or not, two unary expressions exist: `AtEOF` and `InFile`. Even though a typical execution of a `Pablo` program will run until exhausting its input data, the `Pablo` language provides a `TerminateAt` intrinsic to allow a `PabloKernel` to abort early and inform the pipeline that it has fatally terminated. Since caching often bypasses `Pablo` IR generation, such `PabloKernels` must be annotated with an `MayFatallyTerminate` attribute.

`AtEOF` and `InFile`

Suppose the length of the input stream is n. If the value of the $(n+1)^{\texttt{th}}$ bit position of stream S is already 1, `AtEOF`(S) will return a bit stream with only the $(n+1)^{\texttt{th}}$ set. `InFile`(S), on the other hand, will leave any 1-bit set prior to the $(n+1)^{\texttt{th}}$ position.

`TerminateAt`

Given a non-zero bit stream S, `TerminateAt`(S, e) will both trigger a `C++` call-back with the error code e and inform the pipeline that this `PabloKernel` has (fatally terminated). Both this kernel (and potentially its pipeline) will continue their respective iterations of their `DoSegment` loop but no pipeline thread will execute an additional iteration after the signal is observed. Due to multi-threading, discussed in §2.1.5, this may mean up to $num\_of\_threads - 1$ additional segments are executed but no kernel can be invoked once terminated.

A careful eye may note that except for `Lookahead`, `Pablo` intrinsics only move bits 0 or more positions forwards and the `Lookahead` exception requires pipeline intervention to

support it. Any foreseeable intrinsics are expected to have the same restrictions. Because of this inherent lack of "backtracking", we conject that the domain of languages parsable by `Pablo` only slightly exceeds the realm of regular languages, potentially encompassing the class of deterministic context-free languages. The only proven exception to the regular language limitation, however, is the parenthesis matching problem, $\langle \mathrm{S} \rangle \models \langle \mathrm{S} \rangle\langle \mathrm{S} \rangle \mid \texttt{(} \langle \mathrm{S} \rangle \texttt{)} \mid \epsilon$, which we present the pseudocode for the validation thereof in Alg. 5.

Algorithm 5 Validate parenthesis matching

```
1:  procedure ParenthesisMatching(L = [(], R = [)])
2:      Parens ← L ∨ R
3:      PScan ← ScanThru(Advance(L, 1), ¬P))
4:      var Closed ← PScan ∧ R
5:      var Error ← AtEOF(PScan)
6:      var PendingL ← PScan ∧ L
7:      var PendingR ← R ∧ ¬Closed
8:      var InPlay ← PendingL ∨ PendingR
9:      while PendingL ≠ 0̄ do
10:         PScan ← ScanThru(Advance(PendingL, 1), ¬InPlay))
11:         Closed ← Closed ∨ (PScan ∧ R)
12:         Error ← Error ∨ AtEOF(PScan)
13:         PendingL ← PScan ∧ L
14:         PendingR ← R ∧ ¬Closed
15:         InPlay ← PendingL ∨ PendingR
16:     Error ← Error ∨ (R ∧ ¬Closed)
```

Currently, two major issues exist that prohibit us from exploiting such structures further. Firstly, to be able to identify the positions of each parenthesis, we need `Var` **array objects** that dynamically-resize themselves (or for a finite bound to be placed on the nesting depth.) To be useful to the rest of the pipeline, each array object must be an output streamset `Var`. This will require incorporating a notion of an **extensible streamset** into the framework. Earlier versions of the framework supported this but in the move towards treating unmanaged kernel I/O as transferring data through indistinct external streamsets (thereby reducing the number of trivially different copies of the same kernel) this functionality was lost. Secondly, for reasons we discuss in depth in §2.3.4, parsing non-regular languages with `Pablo` is significantly more complicated than regular languages and requires a resizable kernel state object. Support for this exists but is experimental.

⟨Expression⟩ ⊨ ⟨Var⟩ | ⟨Constant⟩ | ... | ⟨IntOrVecBinary⟩
| ⟨IntegerComparison⟩ | ⟨PackingOperation⟩
⟨IntOrVecBinary⟩ ⊨ ⟨IntOrVecBinaryFuncType⟩ ⟨Expression⟩ `,` ⟨Expression⟩
⟨IntOrVecBinaryFuncType⟩ ⊨ `Add` | `Subtract`
⟨IntegerComparison⟩ ⊨ ⟨IntegerComparisonFuncType⟩ ⟨Expression⟩ `,` ⟨Expression⟩
⟨IntegerComparisonFuncType⟩ ⊨ `LessThan` | `Equals`
⟨PackingOperation⟩ ⊨ ⟨PackingOperationFuncType⟩ ⟨Integer⟩ `,` ⟨Expression⟩
⟨PackingOperationFuncType⟩ ⊨ `PackL` | `PackH`

Grammar 2.5: Pablo non-bit stream operation AST BNF grammar

The final sets of operations we cover here are not bit stream based ones nor are they found in typical `Pablo` programs. As mentioned earlier, `Pablo` supports both `Integers` and any uniform field-width input streamset. As implied by Grammar 2.5, these operations are limited but do provide some interesting extensions to the language. Both `Add` and `Subtract` are supported by `Pablo`. When given scalar operands, these perform the expected `Integer`-based arithmetic, which is typically found in index calculation for `Var` array objects; otherwise the same intrinsics will support lane-based vector arithmetic with support from the `IDISABuilder`. Similarly, `Pablo` also provides `LessThan` and `Equals` integer or vector comparisons. Finally, inspired by their inclusion in the **s2$^{\mathbf{k}}$** programming language [52], `Pablo` supports both `PackL` and `PackH` intrinsics. Both operations take two operands, a constant field-width `Integer` and a stream of w-width items. For a given field width k, they will pack the first (resp. last) $k/2$ bits of every adjacent w-sized element. An example is shown in Figure 2.31 wherein $k = w/2$; consequently `PackL` and `PackH` take the even (resp. odd) quartile of each element.

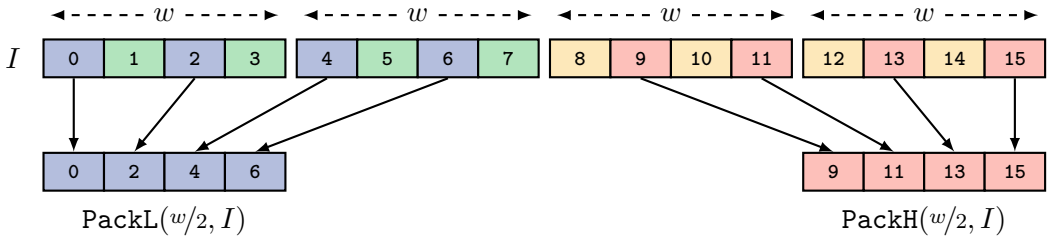


Figure 2.31: Pablo PackL and PackH intrinsics

The primary use case for `PackL` and `PackH` is to perform transposition (§2.2.2) within `Pablo` code. For example, given a RE with a 3-letter prefix such as "`abc`(⟨Complex RE⟩$*$)`d`", it could be more efficient to identify matches at a byte-level prior to transposing. Using an `If` guard and `PackL`/`PackH`, this would be a straightforward process.

In conclusion, when considering the strengths and weaknesses of the `Pablo` language, it is important to consider it within the context of the Parabix framework. Even though `Pablo` does not support backtracking, we can perform some backtracking-like behaviours by constructing a more complicated multi-kernel pipeline. For example, by combining filter by mask (§2.2.3), `Lookahead` and spread by mask (§2.2.4) it is possible to perform variable-length `Lookaheads`. Additionally, with the eventual inclusion of techniques such as *conditional streamset reversal* it may be possible to adapt Sulzmann and Lu's two-stage "forwards-backwards" parse tree construction [107] into Parabix.

2.3.2 BixNum

A `BixNum` is a transposed representation of a non-negative integer sequence. For example, we can represent the 4-bit sequence $\{3, 0, 2, 14, 9\}$ as a 4-stream streamset $S = \{S_0, S_1, S_2, S_3\}$:

S_0	`1...1`
S_1	`1.11.`
S_2	`...1.`
S_3	`...11`

There are a few ways to use `BixNums`: a variety of mathematical operations and equality/inequality comparisons can be generated using the `BixNumCompiler`, resulting in either a bit streamset that is guaranteed to be large enough to hold the resulting calculation or a single bit stream marking the output of each comparison, respectively. Many interesting systems can be developed with this, e.g., transcoding and hashing.

Although UNICODE formats have clear mathematical translations to convert between each encoding format, transcoders involving legacy character sets are often table-driven. Both the `BixNumTableCompiler` and `BixNumRangeTableCompiler` both generate table-based transcoders based on user-defined mappings. The `BixNumTableCompiler` generates `Pablo` code for arbitrary mappings; the `BixNumRangeTableCompiler` optimizes the calculations for the `gb18030` to `UTF` transcoder based on the observation that every codepoint mapping $a_i \rightarrow b_i$ was strictly non-decreasing (i.e., $b_i \geq a_i$ and $b_i > b_{i-1}$.) Moreover, the same mapping function could be applied to long sequences of code points, leading to easily-observable partitions for `If` optimization branches.

Hashing has been a common problem in several Parabix programs [52, 81] but these methods relied on sequential hashing functions to resolve the appropriate `guids`. Recently, the `ztf` compression algorithm has explored bit-parallel hashing methods. Although this methods is still in its infancy and no attempt to determine the hashing strength or collision rates [96], it is not intended to be a cryptographically secure hashing mechanism. Related work in [116] transposed XML element and attribute `guid` integer streams into `BixNum` form to perform XML Schema validation via hand-coded REs `Pablo` programs. This work showed promise but was never fully automated, did not exploit length-grouping (an example

of which is shown in Alg. 4) nor was it ported to the current framework. Combining these approaches is a useful avenue of future work.

2.3.3 Difference between the Pablo block and Pablo builder

For programmers who are unfamiliar with the `Pablo` architecture, there is a common misconception between the intention and use of both the `PabloBlock` and the `PabloBuilder`. The `PabloBlock` is effectively a basic block for `Pablo` programs; it contains a straight-line sequence of statements that will be executed sequentially at run-time. The `PabloBlock` provides a variety of `create` methods, two or more for each instruction in §2.3.1, since every statement can be either implicitly or explicitly named. Implicitly named statements are only given a name when necessary, such as when the `Pablo` IR is emitted to the console. Additionally, no two statements may have the same name. Duplicate names are given a unique suffix id upon name creation, similar to LLVM. Once a `create` method is called, the `PabloBlock` automatically inserts the new instruction after the current insertion point and updates insertion point to be that instruction. This behaviour was inspired by the sequential-construction philosophy of the LLVM `IRBuilder` and provided a way to hide complexities of the internal variable-size memory pool allocator [21]. Like the `IRBuilder`, the insertion point of a `PabloBlock` can be freely changed by the programmer.

Functionally the `PabloBuilder` is nearly-identical to the `PabloBlock` and is equivalent at an API-level. Like the `PabloBlock`, it is intended to generate straight-line sequences of instructions but unlike the `PabloBlock` it retains sole control over the insertion point. This limitation provides an interesting guarantee. Recall that `Pablo` supports both `If` and `While` but does not permit else branches. Because of this restriction, instruction domination is trivially inferable. By maintaining a history of generated instructions, the `PabloBuilder` can automatically perform partial redundancy elimination as it generates code. Additionally `PabloBuilder` is capable of a few minor peephole optimizations such as Boolean algebra simplifications. Even though this mechanism may seem to be both a case of "premature optimization" and redundant in light of the subsequent optimization passes, discussed next in §2.3.4, preliminary studies of both `icgrep` and `editd` showed that `Pablo` IR generation and optimization time can be a significant component of the JIT-compilation time for complex `Parabix` programs and this typically halved that time. However, the `PabloBuilder` is not intended to be interchangeable with a `PabloBlock` and hence they have not been made to inherit from a common interface to help dissuade from this misuse.

2.3.4 Pablo compiler and optimization passes

The `PabloCompiler` is a specialization of the `KernelCompiler` that maps higher-level `Pablo` code to LLVM IR. Compilation here is a non-trivial process because of two factors:

1. `Pablo`'s unbounded bit stream model cannot be verbatimly implemented on commodity hardware. Of note, both `If` and `While` branches are surprisingly nuanced in how they must reasoned about to correctly reflect the unbounded code.

2. As a higher-level domain-specific language, we can optimize `Pablo` IR using techniques that are impractical (and potentially illegal) for general-purpose LLVM code. Moreover, standard LLVM optimization passes typically find few opportunities to improve the SIMD-dominated code generated by the `Pablo` compiler.

Recall that in the unbounded model, every `Pablo` instruction operates on the entire input simultaneously. Within the Parabix pipeline, however, each kernel processes data one segment at a time, already partitioning the unbounded flow of input. Moreover, actual `x86` SIMD instructions (along with similar architectures) are limited *block_width*-sized register operands. When compiling `Pablo` code to LLVM IR, we transform the unbounded model to "block-at-a-time" code wherein segments are subdivided into *block_width*-sized chunks and iterated through in a way that matches the expected flow of data through a streamset.

Conceptually, `Advance`, `ScanThru` and `MatchStar` all rely on addition to initiate their state transitions. Unbounded bit streams can be viewed as arbitrarily-large integers and by chaining a looped sequence of **`ADCX`** (add-with-carry) instructions, we can simulate the addition of such numbers; `SpanUpTo` operates similarly, substituting `ADCX` for a **`SBB`** (subtract-with-borrow) instruction. For simplicity, we refer to all notions of carries and borrows as *carry-flags*, including those without an easily-definable mathematical formulation, such as `IndexedAdvance`. Regrettably, arithmetic within SIMD registers is limited to discrete $\{8, 16, 32, 64\}$-bit lanes that do not have carry-flags prior to `AVX-512` and those for `AVX-512` do not directly support the "horizontal chaining" behaviour required to mimic the unbounded model. Consequently, the primary job of the `Pablo` compiler is to implement carry-flag propagation in software, a task that falls to the appropriately-named **`CarryManager`**. The `CarryManager` is still responsible for reserving a field within the state object of a `Pablo` kernel for every stateful intrinsic and generating the LLVM IR necessary to propagate the carry-flags in the "block-at-a-time" code. The actual code generated for each intrinsic is beyond the scope of this dissertation but can be viewed here[7]. Beyond what was already discussed, there are two key aspects of the `CarryManager` to consider: the `Pablo` kernel state size and the carry-collapsing mode.

Carry state space problem

Every instance of a `Pablo` instruction in the execution trace of a JIT-ed `Pablo` kernel generates its own unique carry that is propagated to the corresponding instance of the same instruction in the subsequent block of the "block-at-a-time" code. That means when

[7] https://cs-git-research.cs.surrey.sfu.ca/cameron/parabix-devel

processing a `While` loop, every instruction and iteration theoretically generates a distinct carry-flag tied to that particular pairing (we discuss when we can avoid this shortly.) Additionally, when testing the condition of any `If` or `While` branch, correctly reproducing the unbounded behaviour means *it is insufficient to only check the condition value*. In the unbounded model, we would enter the branch body and execute every statement within it over the entire input, regardless of whether a particular region of the condition bit stream is non-zero. Replicating this in the "block-at-a-time" code means we must propagate any pending carry-flags even when the current condition "block-value" is zero. To avoid having to iterate through and test every nested carry-flag, a summary carry-flag is associated with every branch and combined with the initial condition "block-value" when determining whether to take the branch. To accommodate iteration-specific `While` loops, at least $\lceil \log_2 n \rceil$ summaries are required to count n pending loop iterations using a half subtractor.

Even ignoring `While` loops, *the number of carry-bits can be excessive* since they tend to increase linearly with size of a `Pablo` program. This is only exacerbated by the fact that we currently have no efficient means to densely pack carry registers to anything smaller than single byte fields. Earlier attempts in improving packing beyond this found the cost of inserting and extracting each carry exceeded its benefit, both due to poor instruction selection by LLVM and increased register pressure due to longer-lived fields. The first problem may be mitigated by later versions of LLVM or by adding specialized intrinsics into the `IDISABuilders` but the latter problem appears to be the result of a fundamental mismatch in the expectation what a typical `BasicBlock` should look like by LLVM and `Pablo`. The standard LLVM register allocators are designed with the assumption `BasicBlocks` will contain straight-line code segments of 10s of instructions opposed to the 1000s of instructions typically found in `PabloCompiler`-generated `BasicBlocks`. Future work on carry-packing should consider evaluating this problem in conjunction with a custom register allocator for a set of `Pablo`-specialized `TargetMachine` subclasses.

Carry-collapsing mode

Although we cannot directly reduce the number of carry-bits required by straight-line `Pablo` code, we can often eliminate the need for iteration-specific carries in `While` loops by means of the carry-collapsing mode. In this mode, we ignore the fact that an instruction is within a `While` loop when assigning its carry field. The $i^{\texttt{th}}$ instruction of a loop will always use the $(\alpha + i)^{\texttt{th}}$ field regardless of the current iteration number, where α is the field number assigned to the carry-producing instruction immediately prior to the loop.

Carry-collapsing mode cannot be used for every loop, however. Suppose a `While` loop executes n iterations. Let $R_{i,j}$ be the bit stream resulting from a stateful intrinsic during the $i^{\texttt{th}}$ iteration of the $j^{\texttt{th}}$ block. Based on the conjecture that the forward progression of bits through a chain of bit stream calculations mirrors the transitions within one or more disjoint parse subtrees of a `LL(1)` parser, we reserve this mode for loops in which we can view each

$R_{i,j}$ as the output of a `FOLLOW` function that parses a set of a mutually-discrete regions w.r.t. other values of i. Specifically, for all $i < n$, $R_{i,j} = R_{(i+1),j}$ or $R_{i,j} \cap R_{(i+1),j} = \emptyset$ and no $R_{i,j}$ values is read outside of the $i^{\texttt{th}}$ iteration of the loop except for $R_{n,j}$, which could be read after the loop by way of an assigned `Var`.[8] Within such loops, the carry-out of any stateful intrinsic (in the "block-at-a-time" code) can have a value of 1 after at most one loop iteration and 0 otherwise. Suppose the carry-out of the $m^{\texttt{th}}$ iteration is 1. Because each $R_{i,j}$ represents mutually-discrete regions, the carry-out from $R_{m,j}$ can can be safely propagated to $R_{1,(j+1)}$ since delaying its propagation will only delay the recognition of a production rule that is disjoint from the region that would otherwise be recognized in $R_{1,(j+1)}$. Consequently, in carry-collapsing mode, each looped instruction may safely consume the disjunction-amalgamated carry-flag generated by executing the $j^{\texttt{th}}$ block when processing the first iteration of the $(j+1)^{\texttt{th}}$ block.

Obviously, this property can hold only when backtracking is not required to correctly parse the input. Thus we conject this optimization is possible only for `While` loops that recognize a regular language. Conversely, *non-regular languages require iteration-specific carries*, which further entails dynamic allocation of carry space since we can only bound the total number of iterations (in the "block-at-a-time" code) for non-backtracking loops. Currently, there is no way of determining whether a `While` loop parses a regular language by analyzing the `Pablo` IR; thus this is another avenue of future work.

Pablo optimization passes

The `Pablo` library offers several optimization passes that were designed with the constraints and intention of the `Pablo` language in mind. Two passes that are executed on every `Pablo` kernel are the `CodeMotion` and `Simplifier` passes. `CodeMotion` performs the standard loop-invariant global code motion but because of `Pablo` only permits `If` and `While` branches, this pass is significantly simpler than the respective LLVM version.

The `Simplifier`, however, combines several concepts: it performs global redundant code elimination, dead code elimination, trivial folding of numerous instructions and intrinsics, including simple Boolean optimizations and reasoning about bit streams known to be constant sequences of all $\overline{0}$s or $\overline{1}$s, as well as strength reduction of a few specific combinations of intrinsics. Because of the simple nesting structure of `PabloBlocks`, we can identify redundant `Vars` by treating their assignments and uses as LLVM stores and loads, applying concepts from both global redundant store and load elimination. Modifying `Vars` tends to expose new optimizations; thus this pass repeats until a "`Var` elimination" fixpoint is found.

The typical `Pablo` program is dominated first by Boolean logic, followed by `Advance` intrinsics. Two passes were created to target each: for Boolean logic, we can apply the

[8]Escaped `Vars` typically return a disjunction of each recognized terminal across all n iterations rather than $R_{n,j}$ itself.

`DistributivePass` to identify instances where we can utilize the distributive law on the Boolean operations. This was done by finding the maximal tricliques using an adaptation of the MICA maximal biclique enumeration algorithm [4]. Although this pass often identified many potential optimizations, aggressively applying it to a fixpoint tended to create long-lived dependency chains that resulted in trading cheap Boolean logic for more expensive register spills and reloads. Moreover this pass has not been updated to reason about or generate `Ternary` operations and thus has not been enabled by default.

Targeting `Advance` operations is less straightforward: we cannot eliminate `Advances` directly but can often combine several into a single call with the `MultiplexingPass`. Multiplexing is a fairly unique optimization of `Pablo` programs. Given three mutually-exclusive bit streams, A, B, and C, we can compress the data into two streams using binary `Or` instructions and then recover the original information using `And` and `Not` operations (Figure 2.32).

Source Data	`--aabacb---abba----c----bcbab----aba--`
$A = [a]$	`..11.1.....1..1...........1.....1.1..`
$B = [b]$	`....1.......11.........1.1.1.....1...`
$C = [c]$	`......1...........1.....1...........`
$D_0 = A \vee C$	`..11.11....1..1....1.....1.1.....1.1..`
$D_1 = B \vee C$	`....1.1.....11.....1....111.1.....1...`
$A' = D_0 \wedge \neg D_1$	`..11.1.....1..1...........1.....1.1..`
$B' = D_1 \wedge \neg D_0$	`....1.......11.........1.1.1.....1...`
$C' = D_0 \wedge D_1$	`......1...........1.....1...........`

Figure 2.32: Example of 3-to-2 Multiplexing / 2-to-3 Demultiplexing

On its own, multiplexing is not significantly useful but if A, B and C are inputs of an `Advance` operation, we can reduce the number of `Advance` instructions in a Parabix program based on the fact that for any integer k the following equations are equivalent:

$$\begin{array}{l|l} A' = \texttt{Advance}(A, k) & D_0 = \texttt{Advance}(A \vee C, k) \\ B' = \texttt{Advance}(B, k) & D_1 = \texttt{Advance}(B \vee C, k) \\ C' = \texttt{Advance}(C, k) & A' = D_0 \wedge \neg D_1 \\ & B' = D_1 \wedge \neg D_0 \\ & C' = D_0 \wedge D_1 \end{array}$$

We are not constrained to only 3-to-2 multiplexing: for every integer $n \geq 2$, there exists a formula for $(2^n - 1)$-to-n multiplexing. Conceptually, the algorithm for multiplexing n variables is similar to one that constructs a truth table of n variables with the all-zero row suppressed. Unlike 3-to-2 multiplexing, some variables can be replaced with a bit stream of all $\overline{0}$s, which is, by definition, mutually exclusive with every bit stream.

Multiplexing, regrettably, is both a costly and an input dependent optimization since we must prove a set of bit streams are mutually exclusive, which currently relies on the Z3 SAT solver. Moreover it is prone to the same long-lived dependency chain problem associated with the `DistributivePass` as well as a lack of support for `Ternary` operations and thus it too has not been enabled by default. Assessment of how best to apply either optimization is an avenue of future work.

Chapter 3

Dataflow programming

Although we introduced the concept of dataflow models in §1.2, in this chapter we focus on a few critical aspects of dataflow programming that the `PipelineCompiler` (Ch. 4) relies upon. We present the relevant theoretical background on synchronous dataflow (SDF) networks in §3.1 and discuss a strategy for offline dynamic storage allocation (DSA) that is tailored for dataflow systems in §3.2. Because the needs of the Parabix framework does not always match the assumptions of typical dataflow programs, we make commentary as necessary to bridge the two worlds.

To review, dataflow networks are modelled as directed multigraphs; each dataflow graph consists of a collection of nodes connected via channels. Each *node* represents a deterministic side-effect-free function and each *channel* is a unidirectional FIFO queue in that transfers a sequence of data from a producing node to a consuming node. Any communication between nodes is strictly through one of its channels. An I/O *port* connects a node to a channel and each channel is between precisely two ports. Each datum, referred to as a *token*, is restricted to a single channel-specific type (e.g., integers, strings, objects, functions, etc). Associated with each port is a *rate* that indicates how many tokens are transferred through the port (i.e., pushed to or popped from the channel) at each invocation of a node. During an initialization phase, channels may be enqueued with initial tokens; these tokens are commonly referred to as *delays*. When every incoming channel to a node has a sufficient number of tokens to satisfy its input rates, a node is considered to be *enabled*. A node may be *invoked* (or *fired*) only after its enabled, at which point it consumes its input tokens and produces any output tokens.

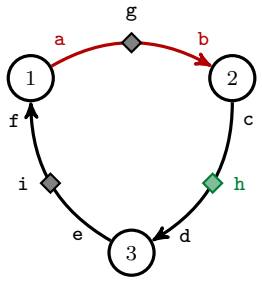
3.1 Synchronous dataflow

The synchronous dataflow (SDF) model, first formalized by Lee and Messerschmitt in 1987 [70, 71], is the prototypical example for static dataflow. Almost all other static models are functionally equivalent to it but have some feature that increases the model's intuitiveness when solving a specific type of problem. It should be noted that only a few Parabix

programs, such as `wc` and `editd`, neatly fit within the SDF model since the majority of them require non-`Fixed` rates. SDF, however, is the most widely explored dataflow model for the purpose of analysis and scheduling. Leveraging the schedulability of SDF networks naturally leads to more efficient pipeline functionality in the Parabix framework.

3.1.1 Constructing a synchronous dataflow graph

To begin we informally define a *schedule* to be any sequence of node invocations for a program P. An *admissible* schedule for P is a schedule in which every invoked node is first enabled. To compute such a schedule, Lee and Messerschmitt construct a SDF graph G, which is a directed multigraph consisting of nodes linked via channels.



(a) SDF graph G

$$\Gamma = \begin{bmatrix} a & -b & 0 \\ 0 & c & -d \\ -f & 0 & e \end{bmatrix} \qquad \mathtt{b}_0 = \begin{bmatrix} g \\ h \\ i \end{bmatrix}$$

(b) Topology matrix Γ and Delay vector $\mathtt{b}_0$ for G

Figure 3.1: Simple example of SDF Graph

Every valid SDF graph can be efficiently analyzed at compile time and in theory can be used to generate an optimal schedule for the depicted program. An example of a SDF graph is shown in Figure 3.1a with nodes indicated by circles and channels by edges. The target of each arc through a channel indicates the consuming node and the letter at both ends of the arc, its I/O rate. The letter by the ◆ indicates the channel delay. From G, a **topology matrix**, $\Gamma = |C| \times |K|$, and a **column vector**, $\mathtt{b}_0 = |C| \times 1$, is computed, shown in Figure 3.1b. Each $(i, j)^{\mathtt{th}}$ entry of Γ indicates the token change of channel i when invoking node j and each row of $\mathtt{b}_0$ is the delay of the $i^{\mathtt{th}}$ channel and $\mathtt{b}_0$ is actually the initial value of a recurrence relationship that describes the current number of unconsumed tokens in each channel. Let $\mathtt{v}_t$ be a 0/1 column vector that has a value 1 in the $i^{\mathtt{th}}$ row if node i is invoked at time t (i.e., the $t^{\mathtt{th}}$ firing vector), then:

$$\mathtt{b}_{t+1} = \mathtt{b}_t + \Gamma \times \mathtt{v}_t \tag{3.1}$$

Suppose G has some sequence of invocations $\Phi = \{\mathtt{v}_1, \mathtt{v}_2, \ldots \mathtt{v}_n\}$ that permits bounded memory and produces the desired output. In other words, at every time t, each row of $\mathtt{b}_t$ is a finite non-negative integer. This implies that Φ must be *periodic*, i.e., if Φ is generated for an infinite sequence of invocations it must be composed of an infinite number of repetitions of the same n-length subsequence. Since Φ is periodic, the starting state of every $m^{\mathtt{th}}$ subsequence is:

$$\mathtt{b}_{m\cdot n} = \mathtt{b}_{(m+1)\cdot n} = \mathtt{b}_0 + m \times \Gamma \times \overrightarrow{\overrightarrow{\mathcal{R}}}, \text{ where } \overrightarrow{\overrightarrow{\mathcal{R}}} = \sum_{i=1}^{n} \mathtt{v}_i \tag{3.2}$$

Eq. 3.2 shows finding the number of times a node must be invoked in an admissible periodic schedule is equivalent to finding a non-trivial column vector $\overrightarrow{\overrightarrow{\mathcal{R}}} = |K| \times 1$ in the nullspace of Γ. Any positive $\overrightarrow{\overrightarrow{\mathcal{R}}}$ s.t. $\Gamma \times \overrightarrow{\overrightarrow{\mathcal{R}}} = \vec{0}$ is a **repetition vector** for G. Any graph G in which a $\overrightarrow{\overrightarrow{\mathcal{R}}}$ can be found is considered *strongly consistent*. Unfortunately consistency does not necessarily entail an admissible schedule exists. For example, we can compute $\overrightarrow{\overrightarrow{\mathcal{R}}}$ for both graphs in Figure 3.2 but we cannot satisfy the input constraints of any node. To prove whether an admissible schedule exists for a G, we must determine whether G is *valid* (Def. 1). Since ① is trivially provable, we focus on ② and ③.

Figure 3.2: Two SDF graphs with no admissible schedule

Definition 1. A SDF graph G is valid if and only if:

① The delay of every channel is non-negative.
② Buffer memory can be bounded.
③ G is deadlock free.

Proving bounded memory

Although finding a repetition vector would prove ②, a cheaper method exists:

Theorem 2. $\exists$ *repetition vector* $\overrightarrow{\overrightarrow{\mathcal{R}}} \implies \mathtt{rank}(\Gamma) = |K| - 1$.

Proof. A non-trivial solution to $\Gamma \times \overrightarrow{\overrightarrow{\mathcal{R}}} = \vec{0}$ exists if and only if $\mathtt{nullity}(\Gamma) > 0$; thus by the rank-nullity theorem, $\mathtt{rank}(\Gamma) < |K|$. We prove $\mathtt{rank}(\Gamma) \geq |K| - 1$ by induction: the respective rank of any single-channel graph G_1 must be 1. Adding a channel to G_1 can only increase its rank if its rates are inconsistent. Suppose Γ_n is the topology matrix of an n-node consistent graph G_n and we construct G_{n+1} by adding a new pendant node to G_n.

$$\Gamma_{n+1} = \left[\begin{array}{c|c} \Gamma_n & \vec{0} \\ \hline \vec{\rho}^{\intercal} & \end{array}\right] \tag{3.3}$$

As shown in Eq. 3.3, for any such n, a new *linearly-independent* row $\vec{\rho}^{\intercal}$ will be added to Γ_n (since it must have a non-zero entry associated with the newly-created pendant node.) Thus adding a new node to a consistent graph can only increase its respective rank but can never decrease it and adding a consistent rate channel cannot increase its rank since it must be a linear combination of the rows already present in Γ_n. [71] □

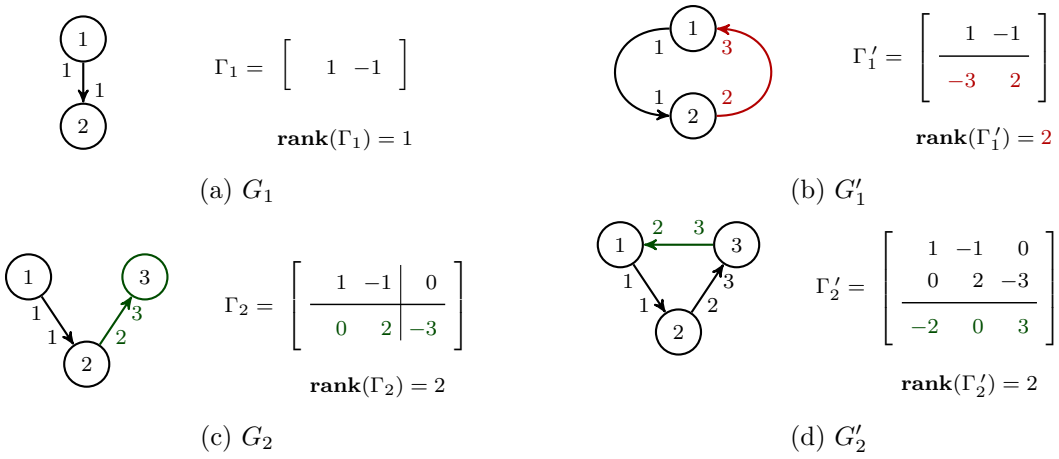


Figure 3.3: SDF graphs with topology matrices

Proving a synchronous dataflow graph is deadlock free

Obtaining $\overrightarrow{\mathcal{R}}$ proves a SDF graph G is valid once it enters its steady state but does not prove that P can enter it (Figure 3.2). The simplest way of proving "liveness" is to compute a *self-timed schedule* [71]. If we assume G is a depiction of a system where each node may fire independently — as is the case for KPNs — every node would be invoked immediately after they were enabled. The key difference between SDF systems and KPNs is that in SDF networks, all I/O rates must be fixed a priori but KPNs do not have that restriction; thus every SDF program has an equivalent KPN. Any sequence of firings in a KPN is an invocation sequence of a self-timed schedule. Since the token history of every KPN channel is deterministic [41, 59], any SDF graph that does not have a self-timed schedule has no deadlock-free schedule [71].

Algorithm 6 Construct admissible self-timed schedule (Adapted from [71])

Require: G is consistent, L contains $\overrightarrow{\mathcal{R}}_i$ instances of each i^{th} node in G and T is initially populated with channel delays.

1: $\Phi \leftarrow \emptyset$
2: **while** $L \neq \emptyset$ **do**
3: **for each** $\alpha \in L$ **do**
4: **if** IsEnabled(α, T) **then**
5: UpdateChannelState(α, T)
6: $L \leftarrow L \setminus \{\alpha\}$
7: $\Phi \leftarrow \Phi \cup \{\alpha\}$
8: **return** Φ

A periodic self-timed schedule Φ is be one in which the number of invocations of each i^{th} node is bounded by $\overrightarrow{\mathcal{R}}_i$. To construct Φ, we generate an arbitrarily-ordered list L consisting of $\overrightarrow{\mathcal{R}}_i$ instances of each i^{th} node and a map T that indicates how many tokens currently

reside in each channel, initially populated by the channel delays. G is deadlock-free if and only if Alg. 6 terminates [71].

Building a minimal repetition vector

Although any repetition vector $\overrightarrow{\mathcal{R}}$ of a valid program can be used to compute an admissible schedule, an arbitrary $\overrightarrow{\mathcal{R}}$ is unlikely to aid in constructing a good one. However, assuming a node cannot be partially executed (e.g., a node that consumes 2 tokens to produce 3 cannot consume 1 to produce $3/2$), it follows any $\overrightarrow{\mathcal{R}}$ we obtain must a multiple of the (smallest positive) coprime integer vector $\overrightarrow{\mathcal{R}}'$, i.e.,

$$\forall\, \overrightarrow{\mathcal{R}} \mid \Gamma \times \overrightarrow{\mathcal{R}} = 0 \;:\; \exists c \mid \Gamma \times \overrightarrow{\mathcal{R}} = c \times \Gamma \times \overrightarrow{\mathcal{R}}' \tag{3.4}$$

While we can calculate $\overrightarrow{\mathcal{R}}'$ from $\overrightarrow{\mathcal{R}}$ by dividing each entry of $\overrightarrow{\mathcal{R}}$ by their common GCD and multiplying the entries of the resulting vector with the common LCM of their denominators, we can avoid calculating $\overrightarrow{\mathcal{R}}$ completely and instead obtain $\overrightarrow{\mathcal{R}}'$ directly by solving a system of linear equations, known as the **balance equation**. Specifically, given a SDF graph $G = (N, C)$ with a topology matrix Γ in which $\texttt{rank}(\Gamma) = |N| - 1$, we can calculate the smallest repetition vector $\overrightarrow{\mathcal{R}}$ of Γ by finding a non-trivial Eq. 3.5, where $\overrightarrow{\mathcal{R}}_i$ is a variable mapped to the $i^{\texttt{th}}$ entry of $\overrightarrow{\mathcal{R}}$, $\texttt{producer}(c)$ and $\texttt{consumer}(c)$ are functions that return the appropriate node label for the channel c, and $\texttt{produced}(c)$ and $\texttt{consumed}(c)$, the rate denoted by the channel c.

$$\bigwedge_{c \,\in\, C} \overrightarrow{\mathcal{R}}_{\texttt{producer}(c)} \times \texttt{produced}(c) = \overrightarrow{\mathcal{R}}_{\texttt{consumer}(c)} \times \texttt{consumed}(c) \tag{3.5}$$

3.1.2 Generating synchronous dataflow schedules for Parabix programs

Given a valid program, Alg. 6 generates an admissible schedule — but not necessarily a good one. Optimal schedule generation is probably the most well-explored topic within SDF literature. In DSP systems, memory is prohibitively expensive so emphasis is often placed on obtaining SDF schedules that minimize memory consumption. Because this is a NP-Complete problem [86], many heuristics have been developed: some favour "maximally-factored" schedules [10–14,53,90,99][1], others employ buffer-sharing strategies to reuse memory [87–89,101], incorporate cache-aware decisions [66], or minimize node "context switches" between invocations when targeting array/vector processors [100]. Although the final class

[1] A factored schedule Φ is sequence of invocations $\{\phi_1, \ldots, \phi_n\}$ wherein each ϕ_i is either a single node invocation or a schedule loop $k\Psi$, where $k \in \mathbb{N}^+$ denotes how many times the nested schedule Ψ is executed. Assuming a SDF graph admits the schedules $\{4A, 2B\}$ and $\{2(2A, B)\}$, the latter would be considered maximally-factored.

does share some commonalities with Parabix, there are two major differences between Parabix programs and traditional dataflow systems that prohibit us from using any of these algorithms:

1. Dataflow channels are FIFO queues, typically abstracting register-to-register datum transfers between microcode fragments. Unlike channels, streamsets often have multiple consumers that may process data at differing granularities and processing rates. Given that streamsets convey large sequences of data each period — often 4 – 64 *KB* of data — naively implementing the dataflow model would require many `memcpy` operations.

An efficient Parabix program ought to implement streamsets as *shared buffers*. Although [87–89, 101] all discussed a notion of shared buffers, sharing there merely meant the sharing of the memory (containers) used to hold tokens but not the tokens themselves. Frameworks that use shared buffers do so to reduce memory requirements but do not factor in their use prior to buffer allocation [16, 32, 49] or do not have an identifiable scheduling policy [106].

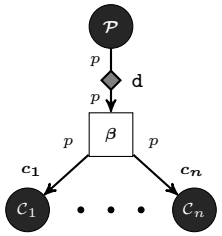
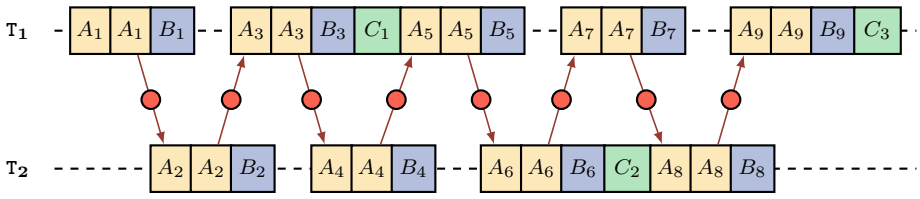


Figure 3.4: Example of a streamset buffer subgraph

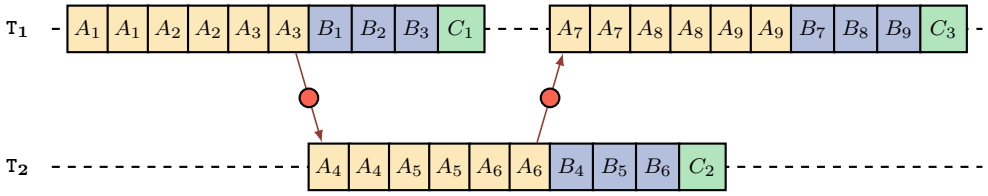
Formally, we represent a Parabix program as a bipartite digraph consisting of kernel and streamset relationships. For example, in Figure 3.4, we have streamset β with a producer $\mathcal{P}$ and consumers $\mathcal{C}_1 \ldots \mathcal{C}_n$. The delay of β is applied to the producer's channel to indicate β is being initially "filled". Each channel is effectively an independent *sliding window* for a shared buffer held by a streamset node β. The processed/produced item counts of each kernel I/O port act as independent read/write pointers. By modelling Parabix programs in such a fashion, we can reason about correctness using the balance equations discussed in §3.1.1 but minimal memory guarantees given for the mentioned scheduling algorithms do not hold for this model.

2. In DSP systems, the cost of calling a function is relatively high compared to the cost of replicating a microcode fragment. Thus SDF compilers historically generated a schedule that adhered to the I/O constraints of the dataflow graph and duplicated the code every node invocation in the schedule. Since fixed loops can be introduced into a DSP circuit with little overhead and can be exploited to reduce total memory costs of a program, factoring schedulers focused on identifying loop-able subsystems and sought to find the one that required the fewest number of code replications with the assumption that such a schedule would also require the least memory [9].

Factoring could be applied to Parabix programs but even if we ignore any I-Cache considerations, factored schedules are also a poor match for Parabix programs. Dataflow nodes are almost-always state-free whereas Parabix kernels are almost-always stateful. This poses no challenge conceptually as state can always be represented with a self-loop in a dataflow graph but poses a significant problem when it comes to multi-threading.



(a) $3(2A, B), 1C$



(b) $6A, 3B, 1C$

Figure 3.5: Impact of looped schedules with stateful kernels

Although several methods of multi-core acceleration were considered for Parabix, the most performant and scalable one on average was found to be the relaxed fixed-data model [75], that assumes a small amount of data will migrate between threads due to non-synchronous I/O. Although it has not been empirically proven, it is questionable whether factored schedules can be as efficient as unfactored ones once the delay imposed by state transference between cores is considered. For example, in Figure 3.5, we depict a two-thread 3 kernel Parabix program in which the number preceding each kernel entry in the schedule indicates the number of strides of work to do per logical segment. Suppose kernel A is stateful; even though (a) has a shorter initial warm-up time on thread 2 compared to (b), it also requires $4x$ of the number of state transfers.

Despite our prior comments, some observations and algorithms for general SDF systems do apply to Parabix. In the following sections, we focus on those that we believe are still applicable to our use case.

Regarding cyclic dataflow schedules

Like its name implies, every cyclic SDF program has a cyclic dataflow graph. Any valid cyclic SDF graph must have a sufficient number of delays along each cycle for the network to enter its "steady state" (e.g., suppose a program P generates an infinite sequence of Fibonacci numbers; without the first two integer tokens, P could never start executing.)

Cyclic graphs are relatively common in DSP systems. However, aside from the implicit self-loop associated with stateful Parabix kernels, *every Parabix program is currently acyclic* — but one case exists that could motivate an exception: `icgrep` supports recursive directory searches, filterable against both file and directory name. Before assessing any file, `icgrep` first collects a list of matching files to process. Currently, this requires repeatedly executing a new file query pipeline for each matching directory but this could be rewritten as a cyclic program with a streamset that is initially filled with the root search path (i.e., a delay token) and repopulated with any newly-discovered directory paths. Thus we do not dismiss the possibility of cyclic programs but instead focus on this as a potential use case.

Generally speaking, schedule generation for cyclic SDF graphs is a NP-Complete problem [86]. Thankfully, when scheduling a cyclic graph every SCC of a SDF graph G can be reasoned about independently [12, 99]. Assuming each contraction of a SCC inserts a valid schedule for that SCC then scheduling the acyclic condensation graph of G always returns a valid schedule for G. It was further observed SCCs can be divided into *trivial* and *non-trivial* classes. Suppose G contains two nodes A and B connected by inverse channels and the delay is such that the data produced by A is not consumed by B within the same period (i.e., a full iteration of a periodic schedule for G). In this situation — for the purpose of scheduling — it does not matter whether A and B have a cyclic relationship [10]. Such relationships are known as **precedence independent**. A channel c is precedence independent whenever Eq. 3.6 holds within a strongly-consistent SDF graph:

$$\texttt{delay}(c) \geq \overrightarrow{\mathcal{R}}_{\texttt{consumer}(c)} \times \texttt{consumed}(c) \tag{3.6}$$

Suppose G is a SCC and P is the set of precedence independent channels of G. Let H be the graph induced by removing P from G. If H is a DAG then G is a trivial SCC and an optimal SDF schedule for H is optimal for G [10]. Acyclic graphs can be reasoned about in polynomial time but optimally scheduling a non-trivial SCC is a NP-Complete problem [86]. Various heuristics [12, 90], minimization optimization algorithms [99] and strategies such as using retiming to introduce cycle-breaking delays [54] have been explored for these more complex cases. Thankfully, non-trivial SCCs are rare in practice [9] and given that the recursive file search use case is guaranteed to be trivial, it seems unlikely that the Parabix framework would require such schedule generation strategies. Thus we leave non-trivial SCCs as a topic for future work and focus on acyclic graphs.

Generating acyclic dataflow schedules

Interestingly, constructing a good schedule for a DAG can still be problematic despite the fact any topological ordering is a valid schedule. Consider that a complete bipartite DAG of $2n$ nodes has $(n!)^2$ topological orderings: even in a Parabix dataflow system, an arbitrary

ordering is unlikely to be a memory-efficient one. For reasons discussed earlier, we focus on a general evolutionary algorithm approach by Zitzler et al. [120]

Evolutionary algorithms are meta-heuristics that model problems as biological evolution. They seeks to iteratively find better solutions via a guided stochastic search whilst avoiding an early convergence to a suboptimal solution. Each potential solution is judged by its relative fitness. An optimal solution is typically one that minimizes or maximizes it, depending on the type of problem being solved. While many evolutionary algorithms exist, Zitzler et al. use the framework shown in Figure 3.6 [120, 121]. Given the preceding discussion, we assume the SDF graph $G = (V, E)$ is a DAG (or a trivially cyclic graph with its precedence-independent channels removed.)

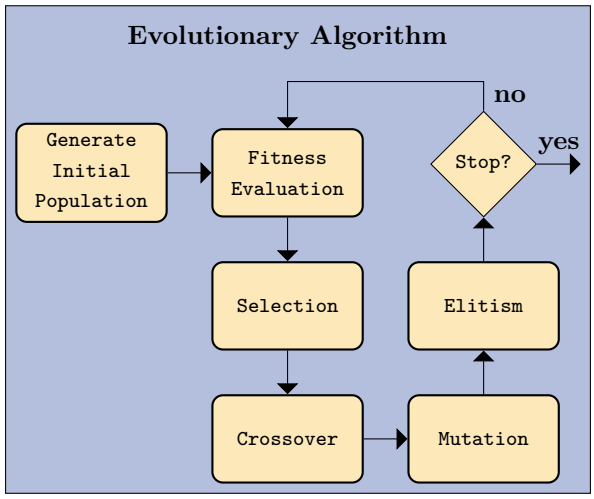


Figure 3.6: Flowchart of the Evolutionary Algorithm (Adapted from [120])

- **Initialization** creates the population $\mathcal{P}$ by randomly generating n initial candidates; a candidate can be any topological ordering of G (including those obtained from the randomized search method detailed in the following section.)[2]

- **Fitness evaluation** evaluates and assigns a fitness score to every new candidate. Both papers by Zitzler et al. use a different metric. In their earlier paper [120], fitness was determined using Murthy's maximally-factored unshared buffer minimization metric [90]. Their latter paper [121] simply sums the maximum number of tokens accumulated within the channels during execution of the candidate schedule. Unfortunately, neither metric are well suited for our use case but an alternate metric that may be more appropriate for Parabix dataflow programs is hinted by Murthy and Bhattacharyya; inspired by the foundational work of Fabri [39], they propose an algorithm for finding a memory-optimal schedule using *offline dynamic storage allocation* [87]. While their work cannot be directly adapted to

[2]in [121], candidates were permutations of V rather than topological orderings; "repairing" of the ordering was instead performed during fitness evaluation.

shared buffers, a related work discussed in §3.2 is general enough to suit our purposes. For now, assume the fittest candidates are those that require the minimal memory.

• **Selection** uses a *binary tournament selection scheme* wherein n pairs of candidates are repeatedly chosen from $\mathcal{P}$ at random and the fittest of the two is moved into the new population $\mathcal{P}'$.

• **Crossover** randomly extracts pairs of individuals, P_1 and P_2, from $\mathcal{P}'$ to create two children C_1 and C_2, inserting them into $\mathcal{P}''$ with crossover probability p_c and P_1 and P_2 otherwise. Since the candidate phenotype is a topological ordering, they utilize an *order-based crossover operator*. As depicted in Figure 3.7, given P_1 and P_2, a random bitstring B is generated and used to create C_1 and C_2 by mixing the orderings of their parents. C_1 is generated by placing each $i^{\texttt{th}}$ node of P_1 into the $i^{\texttt{th}}$ position of its ordering whenever the $i^{\texttt{th}}$ value of B is 1. It then collects all of the unselected nodes, orders them according to P_2 and distributes them linearly into the empty slots. C_2 is similarly generated, selecting initially from the P_2 but choosing its elements when B's value is 0; its missing elements are distributed according to the order of P_1. Finally, the algorithm executes Alg. 6 on C_1 and C_2, returning two topological orderings seeded by those particular permutations.

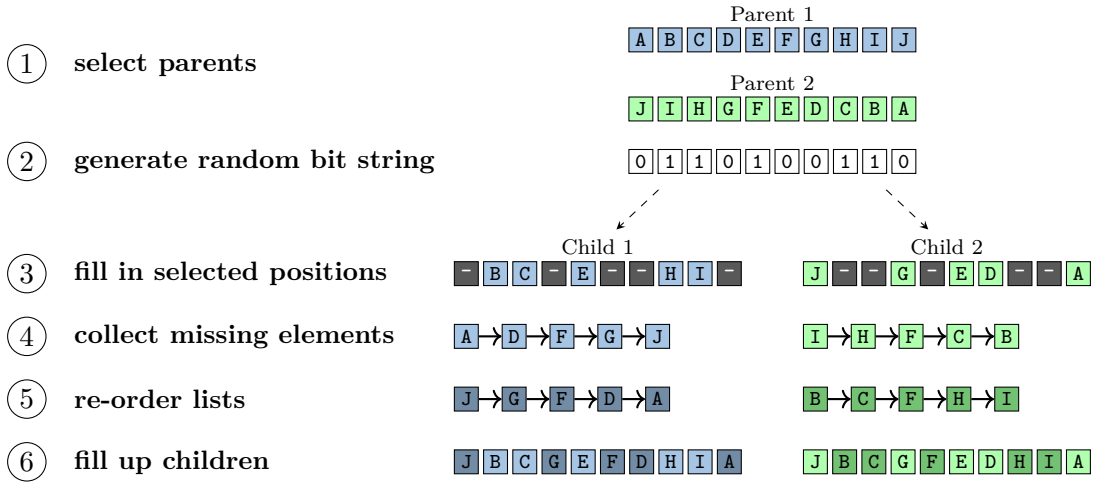


Figure 3.7: Uniform order-based crossover (Adapted from [120])

• **Mutation** has a similar two stage function. Each candidate I in $\mathcal{P}''$ is either mutated with probability p_m or moved into $\mathcal{P}'''$ otherwise. Assuming I is mutated, the algorithm arbitrarily selects two positions in its ordering and randomly permutes the nodes within the indicated sublist (Figure 3.8). Afterwards, the permutation is again "repaired" using Alg. 6 before being inserted into $\mathcal{P}'''$.

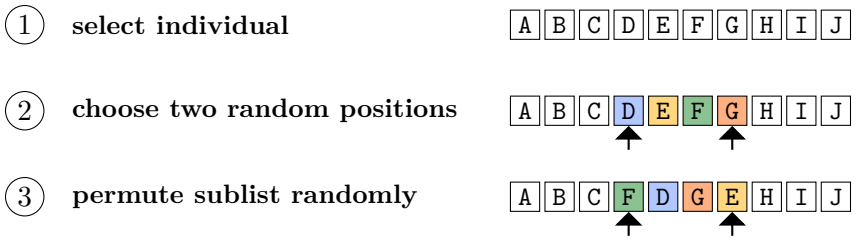


Figure 3.8: Scramble sublist mutation (Adapted from [120])

- **Elitism** arbitrarily replaces one individual from $\mathcal{P}'''$ with the fittest individual of $\mathcal{P}$ [121].

Assuming the algorithm proceeds, it repeats beginning with the evaluation step, substituting $\mathcal{P}'''$ for $\mathcal{P}$.

3.2 Offline dynamic storage allocation for dataflow programs

Dynamic storage allocation (DSA) is one of the fundamental problems in computer science. Given a request for memory, an allocator must return a contiguous section of memory to the program and cannot change its decision once made. While the majority of the literature is interested in servicing online requests [114], we are interested in the more structured problem of offline DSA, where the sequence and duration of requests is known a priori and compile-time decisions can be made regarding memory layout.

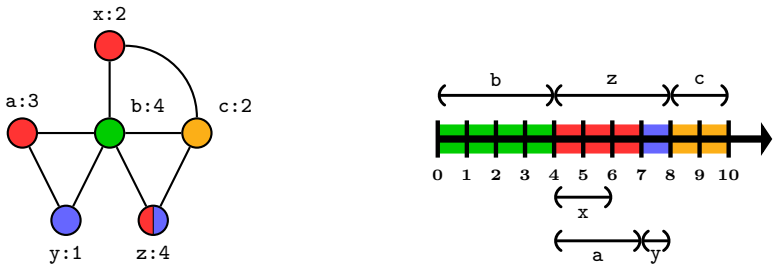
Approaches to this problem are typically based on rectangle packing [19, 45, 46] or weighted interval graph colouring [39, 117]. Here we focus on the latter strategy, specifically Yang et al.'s near-optimal colouring algorithm for dataflow memory layouts. When analyzing the programs designed for the FT64 stream processor, they observed those programs often admitted a schedule in which most of outputs were consumed only by the subsequent node [117, 118] (an observation that matches our own of most current Parabix programs.)[3] A dataflow program strictly composed of such relationships can be represented as a weighted prime comparability graph, which can be optimally coloured in polynomial time [117]. With this observation, they provide a near-optimal heuristic for the problem. We briefly review weighted interval graph colouring, discuss a representation for Parabix programs that is amenable to this problem and then summarize Yang et al.'s work.

3.2.1 Weighted interval graph colouring

Given an undirected graph $G = (V, E)$, graph colouring is the process of assigning hues to each vertex $v \in V$ s.t. no two adjacent vertices have the same hue. Weighted graphs are triples $G = (V, E, \mathcal{W})$, wherein $\mathcal{W} : V \to \mathbb{N}_{\geq 0}$ is a function mapping each vertex to a non-negative weight. A weighted colouring of G can be thought of a function c that assigns

[3]The terminology of [117] and [118] has been altered slightly to better match the Parabix dataflow producer/consumer model.

the hues $[c(v), c(v) + \mathcal{W}(v) - 1]$ to every vertex $v \in V$ s.t. no two adjacent vertices share the same hues. This is often visualized by mapping every $v \in V$ to a real line of width $\mathcal{W}(v)$, such as shown in Figure 3.9 (grouped for visual simplicity.)



(a) (b)

Figure 3.9: Weighted interval graph (a) and corresponding interval colouring (b) (Adapted from [117])

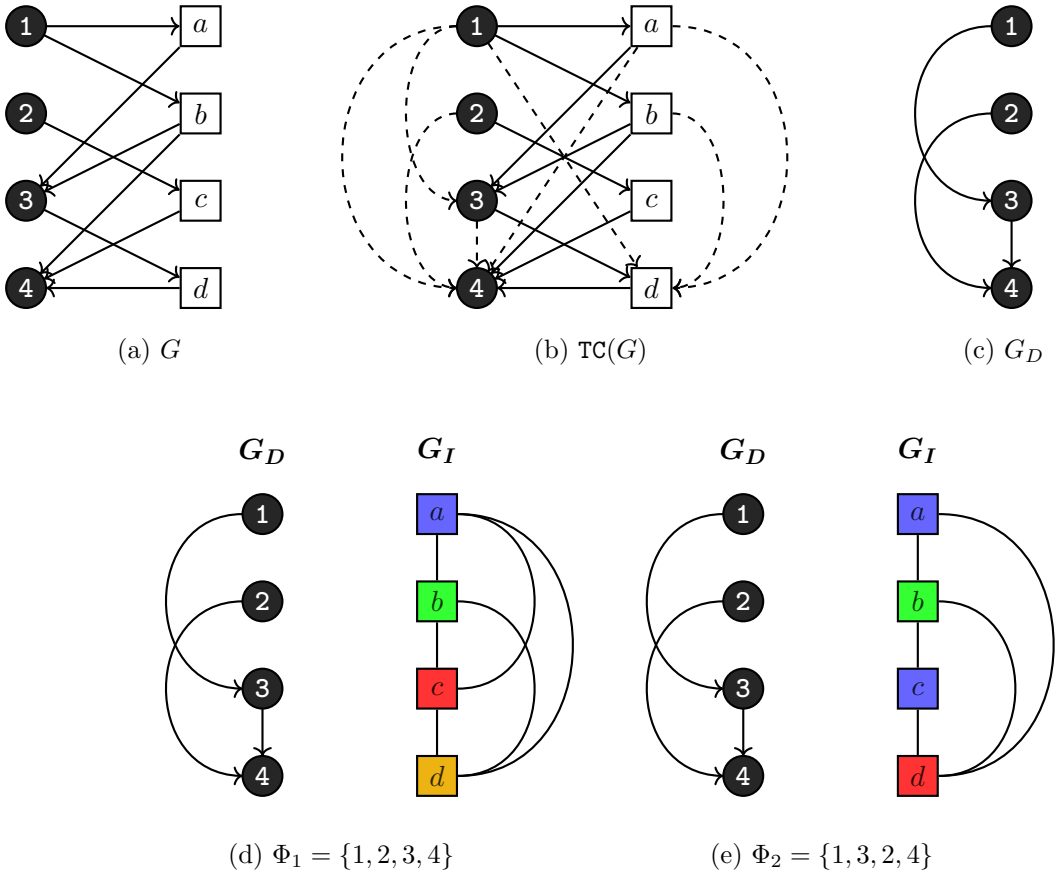
Using weighted interval graphs to model memory allocation is fairly intuitive: each hue marks a unit of contiguous memory (e.g., a byte) allocated to the corresponding buffer. An optimal colouring of G directly indicates the minimum amount of memory required by a program and provides a memory partitioning strategy necessary to achieve it. Here an optimal colouring indicates one that uses the fewest number of hues required to properly colour G. This is known as its *chromatic number* and is denoted by $\chi(G, \mathcal{W})$.

3.2.2 Representing Parabix schedule generation as a DSA problem

Suppose the vertices of a Parabix dataflow graph G are partitioned into disjoint sets K and S containing the kernel and streamset nodes of G, respectively. Let the kernel dependency graph G_D be a transitive reduction of the induced subgraph generated by computing the transitive closure of G then removing S. An example of this process can be seen in (a) – (c) of Figure 3.10, where we begin with dataflow graph of kernels 1 – 4 and streamsets a – d and eventually compute G_D. Given a schedule Φ that is admissible for G_D, its trivial to compute a streamset interval graph G_I by tracking the use-def relationships in G. Each node in G_I represents a streamset in S and an edge exists between two nodes in G_I if and only if both streamsets are mutually alive at any stage of Φ. The life span of a streamset is inclusively between its production and its last consumption w.r.t. Φ. Subfigures (d) and (e) show the two possible schedules of our example graph and the resulting coloured interval graphs computed by this process.

Scheduling the dependency graph G_D using any dataflow scheduling algorithm will return an admissible schedule for G but to leverage G_I as a cost metric, we need a scheduling algorithm capable of utilizing it. The evolutionary algorithm discussed in §3.1.2 fits our criteria: by setting the fitness objective to minimize $\chi(G_I, \mathcal{W}_S)$, where $\mathcal{W}_S$ indicates the

required capacity of each streamset we will naturally converge on a schedule that requires minimal memory.



(a) G (b) $\texttt{TC}(G)$ (c) G_D

(d) $\Phi_1 = \{1, 2, 3, 4\}$ (e) $\Phi_2 = \{1, 3, 2, 4\}$

Figure 3.10: Example of the transformation of a simplified Parabix dataflow graph to dependency and interval graph

It should be noted, however, that "delayed" channels cannot be easily shared [87–89] and variable rates constitute a form of variable delay. To keep the problem tractable, every known buffer-sharing approach utilizes a "coarse-grained" sharing model wherein a channel must be provably empty at a particular point in Φ before its memory can be reused. Since sharing channels is trivially $\leq_P$ sharing streamsets, we assume this is equally true of Parabix streamsets. Correspondingly, each "delayed" streamset must be assigned a unique colour or preferably suppressed from G_I as those colours would represent a common lower-bound.

3.2.3 Yang et al.'s colouring algorithm

Based on the observation that dataflow programs often admitted a schedule in which most of outputs were immediately consumed, Yang et al. presented a novel graph colouring algorithm tailored for this use case [117]. We begin by defining some necessary terminology.

Given an connected undirected graph G, an *orientation* of G is a function α that assigns a direction to each edge. Let G_α be the digraph generated by applying α to G; α is an

acyclic orientation if and only if G_α is a DAG. A good colouring of G can often be obtained via a first-fit colouring of any depth-first ordering of G_α but to optimally colour G, we must consider all possible acyclic orientations. Specifically, $\chi(G, \mathcal{W})$ is equal to the smallest weight of the heaviest path through the acyclic orientations of G [117]. Let $\mathcal{A}(G)$ be the set of acyclic orientations of G and $\mathcal{P}(\alpha)$ be the set of directed paths in α:

$$\chi(G, \mathcal{W}) = \min_{\alpha \in \mathcal{A}(G)} \max_{\mu \in \mathcal{P}(\alpha)} \mathcal{W}(\mu) \tag{3.7}$$

Computing $\mathcal{A}(G)$ is a potentially daunting task but given their observation, Yang et al. reasoned that most dataflow programs could be decomposed into a set of comparability graphs [117][4]. Formally defined in Def. 4, every comparability graph has a transitive orientation and the type of comparability graph we consider here is the one in which a first-fit colouring induced by a transitive orientation is optimal.

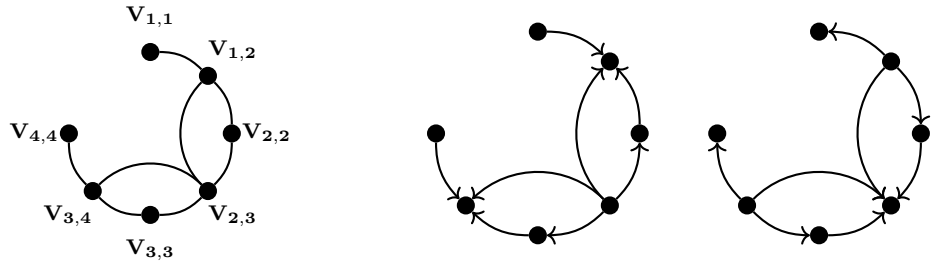
Definition 3 (Greedy first-fit colouring algorithm). Let C_v be the hues assigned to vertex v. A first-fit colouring accepts a vertex list $\mathcal{L} = \{v_1, v_2, \ldots, v_n\}$ and sequentially assigns each $v_i \in \mathcal{L}$ the colours $C_{v_i} = [k, k + \mathcal{W}(v_i) - 1]$ s.t. k is the smallest positive integer in which for every adjacent vertex $v_j \prec v_i$, $C_{v_i} \cap C_{v_j} = \emptyset$.

Definition 4 (Comparability graph). An acyclic orientation α of an undirected graph G is *transitive* if and only if $(x, z) \in G_\alpha$ whenever $\{(x, y), (y, z)\} \in G_\alpha$. G is a comparability graph if and only if it can be transitively orientated. Comparability graphs are super perfect graphs, meaning that their weighted chromatic number is equal to the weight of their heaviest clique. A comparability graph G is a prime comparability graph if and only if it is uniquely partially orderable (UPO), which which is equivalent to saying G has precisely two transitive orientations, wherein each reverses the edge orientation of the other. A prime comparability graph has precisely one χ colour class. [47]

Yang et al.'s main theorem is as follows: given a dataflow graph G_D with m nodes, the corresponding interval graph G_I can be partitioned into $2m - 1$ (potentially empty) independent vertex sets: $S_{1,1}$, $S_{1,2}$, $S_{2,2}$, $S_{2,3}$, …, $S_{(m-1),m}$, $S_{m,m}$, where each $S_{i,j}$ is the set of streamset nodes that are live only when invoking kernel i and j. Let $\mathcal{G}(S_{i,j})$ be the subgraph of G_I induced by $S_{i,j}$. By definition, each $\mathcal{G}(S_{i,j})$ is a clique; therefore any permutation of $S_{i,j}$ admits a transitive orientation and $\mathcal{W}(S_{i,j}) = \sum_{v \in S_{i,j}} \mathcal{W}(v)$.

Let $H = G_0\,[\mathcal{G}(S_{1,1}), \mathcal{G}(S_{1,2}), \ldots, \mathcal{G}(S_{m,m})]$. H is a comparability graph if and only if G_0 is a comparability graph, which is the graph generated by contracting each disjoint $\mathcal{G}(S_{i,j})$ subgraph [47]. Figure 3.11 illustrates the structure of G_0 when $m = 4$. By construction, G_0 is UPO and its two transitive orientations are forced by orienting any edge.

[4]Yang et al. provide two theorems: one for when the number of invocations in Φ is even and another when no cyclic relationships exist in G. Given the discussion regarding cyclic dataflows in §3.1.2, we consider only the latter theorem. Specifically, 4 and 4.3 of [117] and [118], respectively.



(a) G_0 (b) Transitive orientations of G_0

Figure 3.11: Example of G_0 when $m = 4$ (Adapted from [117])

Obviously some streamsets will not be solely consumed by the subsequent invocation. Suppose when given a fixed schedule Φ, we can identify the live range of the streamset nodes of $G_I = (V, E)$ based on the maximum distance of the producing and consuming kernel(s) w.r.t. Φ. Yang et al. observed that a near-optimal solution can often be obtained by partitioning V into two independent subsets [117]. Specifically,

$$\mathcal{V}_1 = \{v \in V \mid v\text{'s live range is at most one.}\}$$
$$\mathcal{V}_2 = \{v \in V \mid v\text{'s live range is two or more.}\}$$

We have already addressed $\mathcal{V}_1$ since $\mathcal{V}_1$ is a superset of each $S_{i,j}$ vertex sets. Although G_0 must be a comparability graph, $\mathcal{G}(\mathcal{V}_2)$ is not necessarily one. Yang et al. suggest, however, that due to the rarity of "long-lived" channels, $\mathcal{G}(\mathcal{V}_2)$ is almost always a forest [117]. Every tree is a bipartite graph, observable by partitioning its vertices into even-level and odd-level sets, A and B. Every bipartite graph has exactly two transitive orientations, obtained by orienting all edges from A to B or v.v. [47]. Therefore any forest F is a comparability graph with $2^{|\mathrm{CC}(F)|}$ transitive orientations, where $\mathrm{CC}(F)$ is the set of connected components of graph F. Suppose $\mathcal{G}(\mathcal{V}_2)$ is not a forest: to determine an acyclic orientation of $\mathcal{G}(\mathcal{V}_2)$, Yang et al. claim it suffices to use its *maximal spanning forest*[5] [118]. We improve upon their research in Appx. A.

Let us return to the original problem with the simplifying assumption that both G_0 and $\mathcal{G}(\mathcal{V}_2)$ are both UPO. To complete the colouring process, we must combine both G_0 and $\mathcal{G}(\mathcal{V}_2)$. Let G'_I be the union of both graphs. Although we can optimally colour G_0 and $\mathcal{G}(\mathcal{V}_2)$ independently, $\mathcal{G}(\mathcal{V}_1 \cup \mathcal{V}_2)$ likely contains edges that cross $\mathcal{V}_1$ and $\mathcal{V}_2$. In other words, we have yet to reason about:

$$\mathcal{E} = \{xy \in E \mid x \in \mathcal{V}_1, y \in \mathcal{V}_2\}$$

[5]NOTE: Yang et al. use the term maximum spanning forest but since there is no discussion of edge weights or line graphs in [117] or [118], we assume they mean a maximal (i.e., full) spanning forest, which is a spanning tree that retains the same connectivity as the original graph.

The edges in $\mathcal{E}$ must be accounted for in any proper colouring of G'_I. Let $\mathcal{T}(G)$ be the set of transitive orientations of some graph G. In Eq. 3.7, we stated the problem of calculating $\chi(G, \mathcal{W})$ was to find the smallest heaviest path through the acyclic orientations of some acyclicly-orientable graph G. Every acyclic orientation $\alpha \in \mathcal{A}(G'_I)$ that induces an optimal colouring of G'_I will orient its edges according to one of the orientations of $\mathcal{T}(G_0 \cup \mathcal{G}(\mathcal{V}_2))$ [117]. Consequently, for possible every orientation induced by $\mathcal{T}(G_0) \times \mathcal{T}(\mathcal{G}(\mathcal{V}_2))$, a unique orientation of $\mathcal{E}$ must be found to determine an optimal colouring of G'_I. Note, given the $G_0[\ldots]$ decomposition of $\mathcal{G}(\mathcal{V}_1)$, we know there are at least $N = 2^{|\mathrm{CC}(\mathcal{G}(\mathcal{V}_2))|+1}$ useful orientations to consider. However we do not necessarily need to reason about all $2^{|\mathcal{E}|} \times N$ orientations. Recall that $\mathcal{G}_\alpha(\mathcal{V}_2)$ is a bipartite digraph. Every $y \in \mathcal{V}_2$ must be a source or sink w.r.t. $\mathcal{G}_\alpha(\mathcal{V}_2)$. Yang et al. guarantee an acyclic orientation by orienting each edge $xy \in \mathcal{E}$ from $y \to x$ or $x \to y$ whenever y is a source or sink, respectively [117]. The minimal weight orientation of these will not necessarily lead to an optimal colouring but assuming $\mathcal{G}(\mathcal{V}_2)$ is naturally a forest, the colouring can only be worse by at most $\mathcal{W}(u) + \mathcal{W}(v)$ where both u and v are differing nodes of the greatest weight in G'_I [118]. Yang et al. contend N ought to be very small and therefore a brute-force enumeration is reasonable [117].

3.3 Related works

Given that the problem of dataflow networks has been actively studied for over 50 years, even when we limit ourselves to considering only the recent frameworks that support variable rates, it is unsurprising that many frameworks supporting it exist. These frameworks tend to fall into one of two categories: simulators (e.g. C-HEAP [92] and Ptolemy II [38]) and interpreters/compilers (e.g., DAL [105], Fastflow [2], PRUNE [17], RaftLib [8], SHIM [37], StreamDrive [106], and StreamIt [48, 110].) Of the latter category, these frameworks are all `C++` libraries that provide new capabilities to the standard `C++` architecture. To support variable rate dataflow, they model and implement programs as BDFs, KPNs or reactive process networks (RPNs) (an extension of the KPNs model.) Of the frameworks listed, the only one that utilizes static scheduling (instead of work-queue-based dynamic scheduling) is the StreamIt library. StreamIt, however, only supports SISO kernels (i.e., filters) and variable rate production rates [29]. Thus the framework described in this thesis is unique in how it supports variable rate dataflow with a static linear-pipeline but unlike many is also restricted to acyclic dataflow programs.

Chapter 4

Pipeline compiler

The `Pipeline` kernel is the backbone of every program generated by the Parabix framework; it is responsible for handling multi-threaded synchronization, the majority of memory management concerns and invoking kernels in some logical order. To reliably provide a near-linear multi-threaded speedup for up to $\lfloor {}^{total\ single-threaded\ time}/_{\max(total\ kernel\ time)} \rfloor$ threads, every decision regarding the construction of the `Pipeline` must be carefully considered. However, despite its importance, a `Pipeline` is a standard cachable kernel that — in theory — is JIT-compiled once for the lifetime of each particular executable. The task of compiling a `Pipeline` according to the programmer specifications falls to the `PipelineCompiler`, which was the focus of the development work for this dissertation.

Programmers construct Parabix programs by instantiating kernel and streamset objects by interacting with the `PipelineBuilder` class. The relationships between these objects are provided to the `PipelineCompiler` in the form of a program graph, detailing the I/O relationships between the objects. For the purpose of caching, a serialized string representation of this graph is the default name of the `Pipeline` object. The compiler itself is divided into discrete analysis and code generation phases. This chapter addresses on the analysis phase as it encapsulates the theoretical contributions of this dissertation. The analysis phase itself is divided into a sequence of subphases; the major subphases include:

1. Conversion of program graph to internal compilation graph
2. Preliminary SDF-partitioning pass
3. Computation of repetition vectors for each SDF-partition
4. Variable-rate simulation of inter-partition communication
5. Final partition identification
6. EA-based scheduling of both intra and inter partition kernels
7. Buffer layout

4.1 Conversion of program graph to internal graph

Upon construction of the `Pipeline`, the `PipelineBuilder` provides it with a bipartite program graph, detailing the kernel and streamset relationships. This graph instructs the `PipelineCompiler` as to what type of program it is generating but several transformations are possible prior to compilation.

1. The identification and removal of redundant and/or unnecessary kernels from the program. In the internal graph, the `Pipeline` object under compilation is represented by two vertices $\mathcal{P}_{in}$ and $\mathcal{P}_{out}$ where the streamsets and scalars produced by $\mathcal{P}_{in}$ are the inputs to the `Pipeline` and the streamsets and scalars consumed by $\mathcal{P}_{out}$ are its outputs. A kernel is considered useful if it exists on any $\mathcal{P}_{in} \rightsquigarrow \mathcal{P}_{out}$ path or is explicitly marked as `SideEffecting`. Additionally, every kernel not annotated with a `Family` or as `SideEffecting` attribute is considered to be deterministic. Since the inputs to any deterministic function defines its outputs, all but one instantiations of the same kernel class with identical inputs are removed from the internal graph and the appropriate I/O relationships are remapped. As of now, all Parabix programs have acyclic program graphs and thus these identifications are made according to the topological ordering of the internal graph; this process can be easily modified to execute to a fix-point in cyclic programs.

2. `PopCount` processing rates are invisibly transformed into **partial sum** relationships. This requires inserting a new `PartialSum` kernel and modifying the I/O relationships to suit in the internal graph. Each $k^{\texttt{th}}$-value of `PartialSum` streamset indicates the number of 1-bits present in the `PopCount` reference streamset within a given span. This span-length is the GCD of all of the kernel stride lengths with a `PopCount` port associated with the same reference streamset.

4.2 Partitioning of Parabix programs into synchronous subgraphs

Each thread of a Parabix pipeline invokes its kernels according to a predefined sequence, repeating until all external input is exhausted. Each time a kernel is invoked, it processes one logical segment of data. The objective of partitioning is to reduce the number of instructions within the `PipelineKernel` by identifying groups of kernels in which the length of the $k^{\texttt{th}}$ segment of any kernel in the group has a linear relation with every $k^{\texttt{th}}$ segment length of the other kernels within the group. In other words, every partition group is independent SDF program that can be reasoned about using the concepts and equations discussed in §3.1. Because of this, we can statically model the flow of information through its kernels. This eliminates the need for testing any I/O requirements of any streamset that is strictly local to that particular partition. Moreover, based on the discussion in §3.1.1, it is trivial to

prove that the memory for any partition-local streamset is both bounded and thread-local, which the memory management system within the pipeline can take advantage of.

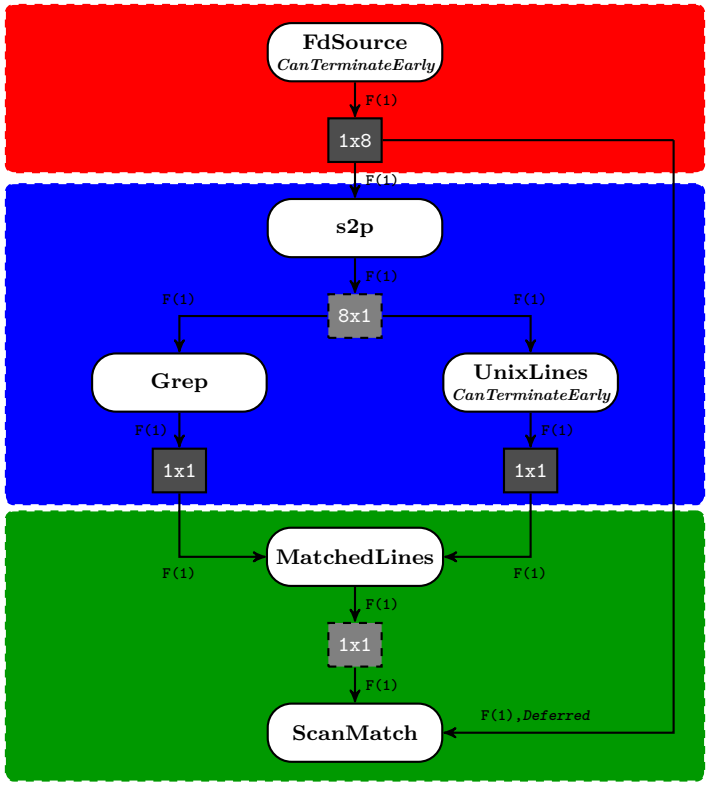


Figure 4.1: Simplified example of partitioned `icgrep` program

Our goal is to identify the fewest number of partitions in which a linear relation between all kernels within each partition is guaranteed to exist. The objective of our approach is identical to that of Gu et al.'s [50]. However, their method assumes that the programmer cannot mix fixed and variable rate dataflow between nodes. Instead, our approach was inspired by work in static taint analysis but tailored for our use case — specifically our notion of processing rates and attributes. For example, Figure 4.1 provides a simplified example of an `icgrep count-only` pipeline in which the kernels within it are divided into three partitions. Even though all of the I/O within the program is fixed-rate, the `CanTerminateEarly` attributes indicate a potential change of information flow that could invalidate the linear relationships. `Bounded` and `PartialSum` rates can have a similar effect as can other attributes.

4.2.1 Identifying partitions

When clustering kernels within partitions, we must consider both the logical requirements and the run-time behaviour imposed on the program with such a grouping. As discussed in §3.1.1, I/O rates within a SDF graph is represented with a topology matrix Γ and for

every consistent SDF graph, $\texttt{rank}(\Gamma) = |K| - 1$ [71]. An obvious corollary to this is that every consistent SDF topology matrix has precisely one free variable. Intuitively, this free variable represents the common symbolic scaling factor for the segment length of the kernels within the same partition. At run-time, this length is determined by both the production rates of the preceding partitions and the consumption rates of the kernels within the current partition. To maximize throughput, we want to determine the largest possible scaling factor at run-time — but multi-threading is a significant complication.

The performance of any pipelined program is dependent on the execution time of the most time-consuming stage. Synchronization in Parabix guards each kernel invocation; although this prevents non-deterministic behaviour, it only ensures that we know the current state of the kernel whose lock we acquired; a kernel's knowledge of I/O state of other kernels is often delayed. For any $i \leq j$, the $i^{\texttt{th}}$ kernel may acquire the $j^{\texttt{th}}$ kernel's lock but doing so would increase the size of the pipeline stage to encompass the kernels between i and j. Even if it is possible to increase the size of a partition by acquiring the $j^{\texttt{th}}$ lock, this could drastically decrease performance. Consequently, we limit each partition to a single "**root**" kernel whose input solely decides the segment lengths of the kernels within its partition. Thus the segment length scaling factor it deduces cannot cause any kernel within the partition to attempt to process more data than it can read. In other words, some property associated with the I/O rates of a root entails that some change in the dataflow may occur when transitioning between from the kernel invoked prior to the root and after. Thus the process of identifying partitions is equivalent to finding such roots in the program and determining what kernels are dominated by it and the roots that strictly dominate their root.

Partitioning occurs in two stages: In the first stage, we have only the internal graph as information; the goal is to conservatively identify a set of SDF partitions. This stage is technically optional but helps improve the efficiency of the simulator used during the dataflow analysis phases. The second stage determines the final set of partitions and uses the information provided by the simulator to reason about more subtle interactions between stride-length, `PartialSum` rates[1] and `LookAhead` and `Delay` attributes.

4.2.2 Preliminary SDF-partitioning pass

1. We begin with the internal graph G consisting of sets of kernel and streamset nodes connected by channel edges annotated with I/O processing rates and attributes. Each node associated with an unbounded bitset.

2. Given an arbitrary topological ordering of G, we traverse G's nodes and initialize the bitset of each vertex v to be the disjunction of the bitsets of v's incoming channels.

[1] `PartialSum` can also be modelled using the VRDF model as the symbolic rates discussed in [113] has a one-to-one correspondence with a reference streamset but the algorithm provided in [113] fails to expose all of the opportunities in our programs.

3. If v is a kernel node with an in-degree of 0, its bitset is flagged with a new marker (that has not been previously associated with any node) because we cannot conservatively assume that the relative output lengths of two arbitrary sources will be equivalent.

4. If any incoming channel of a kernel node has a non-`Fixed` rate, a `LookAhead` or `BlockSize` attribute, we flag v's bitset with a new marker since some potential change of data flow is possible at run-time.

5. After all incoming channels have been processed, we propagate the value of v's bitset through its outgoing channels. If the output port has a non-`Fixed` rate or has a `Delayed`, `Deferred` or `BlockSize` attributes, we add a new bit marker to the target of the outgoing channel's bitset. Note that we do not consider `Add` or `RoundUpTo` attributes despite the fact they alter the dataflow during final-mode processing. Kernels rely on the partition root to inform them that they are entering this phase but once they do, they will completely process their inputs regardless of the segment length of the root.

6. After completing the traversal, we generate a subgraph for each set of kernel nodes in G with an identical bitset. Each weakly-connected component within these subgraphs is marked as belonging to the same synchronous partition. The reason we isolate each component is because our next phase requires us to solve the balance equations for each partition to determine the data flow between intra-partition kernels. However, we are only guaranteed that some integer relationship exists between the amount of data flowing into each component but deducing this factor requires solving the balance equations for all previous partitions. This effectively serializes the work performed by the solver.

4.2.3 Variable-rate simulator

The simulator empirically estimates the how much data flows through at run-time each segment (i.e., pipeline time step.) The quality of this estimate depends on the accuracy of the `ProcessingRateDistributionModel`, assigned by the programmer. We are primarily interested deducing in the mean-value and standard deviation of common scaling factor associated with each partition. However, because the sometimes subtle interactions between `PartialSum` rates, stride lengths and attributes, multiple partitions may actually be synchronous despite not sharing an immediate predecessor. The secondary goal of this phase is to identify such relationships and report to the subsequent phases whether the partitions can be safely contracted in the final program. *This section is for information purposes only and is not critical to the understanding the remainder of this section.*

Generating the simulator is not particularly complex. With the information provided by the preliminary partitioning phase, a graph is constructed in which every kernel belonging to the same partition and partition-local streamset is contracted into a single vertex. Nodes

are then added for each inter-partition streamset, which are connected by edges to the appropriate partition nodes. These edges are annotated with the appropriate processing rate and attribute information, scaled by the value of their repetition vector of their producer/consumer. The only complications of note are in how `PartialSum` rates and `BlockSize` attributes are treated. While `Bounded` rates simply generate a random number within their bounds, `PartialSum` rates retain a history within a circular buffer to ensure that every `PartialSum` port with the same reference streamset will read the same sequence of values. Since kernels can have differing stride lengths and repetition values, the values within the circular buffer are stored as a partial sum to allow for differing step sizes. Output ports with a `BlockSize` attributes are particularly interesting. Ignoring the data arrangement implications, kernels produce data at some rate but only release it for use at some fixed rate. Streamset nodes are annotated with the producer's `BlockSize` value and will only release data as appropriate during the simulation.

After the graph is constructed, we simplify it with the following transformations:

1. Any streamset node incident to only `Fixed`-rate ports and not influenced by a `Delay`, `LookAhead` or `BlockSize` attribute, is normalized by dividing the number of items transferred per invocation by the GCD of their rates.

2. The streamset node of any equivalent `Fixed`-rate output is contracted into a single vertex.

3. Edges corresponding to equivalent countable-rate inputs originating from the same streamset are merged. We ignore `LookAhead` attributes when assessing equivalence and retain only the largest `LookAhead` value in the merged edge. Although unlikely to occur in practice, any edge with an equivalent `LookAhead` amount, we need only consider the largest `Fixed`-rate or `PartialSum` step size.

4. We apply a transitive-reduction-like pass to the `Fixed`-rate inputs of a streamset node. For any streamset with multiple `Fixed`-rate consumers, we determine the reachability of said consumers and identify whether it is trivially guaranteed to have a sufficient number of items.

5. Any streamset node with exactly one `Fixed`-rate producer and `Fixed`-rate consumer that is not annotated with a `Delay`, `LookAhead` or `BlockSize` attribute, is edge-contracted with its producing partition.

The simulator itself is effectively a KPN [59] but since the items queued within each channel are fungible, the simulator also shares some superficial properties with Petri-nets [95]. We treat both `LookAhead` input attributes and `Delay` output attributes as delays in the network. Unlike KPNs, however, rather than forcing delays to be satisfied by a prelude phase, we record them as a negative number of items in the appropriate channel and force the first iteration of the simulator to satisfy them.

When running the simulator, we begin with a brief **greedy demand-driven execution**. In this mode, nodes are invoked in a reverse-topological order, back-propagating "demands" (a negative number of items) through the network. Each sink must be invoked at least once per time step. Preceding partition nodes must be invoked a sufficient number of times that they produce enough data to satisfy all of their outgoing channel demands. Nodes in a greedy demand-driven network will continue executing even after they have satisfied their demands so long as there is sufficient available input. The purpose of this phase is to determine how much input is required from each source kernel and in the cases of having multiple sources, an approximate ratio between their relative firing rates.

The next step is to execute a **data-driven simulation**, wherein kernels are invoked in a topological order as many times as possible for their given input. We use the average number of strides-per-segment of the source kernels from the demand-driven simulation to dictate their firing rate during the data-driven one. However, since the program accepts a user-defined *segment-length*, we scale the firing rate of the source with the smallest strides-per-segment to be as close as possible to the user-defined length but we take into account the immediate consumers (unreachable by any preceding consumer) and ensure that they are able to execute at least one stride per time step regardless of the segment-length value.

In this simulation mode, we compute the average strides-per-segment and its standard deviation of each kernel, which will later be used by both the scheduler and memory layout processes. As mentioned, some partitions can be fused into a single synchronous unit. The secondary goal of this stage is to identify such subgraphs, which we refer to as **linked partitions**. On the first iteration of the data-driven simulator, we construct a complete undirected graph G where each vertex represents one of the partition nodes. Given some canonical ordering $a \prec b$, each (a, b) edge indicates the ratio r between the number of strides executed by partitions a and b. If at any subsequent time step, $\texttt{strides}_a : \texttt{strides}_b \neq r$, we remove the edge from G. At the end of the simulation, the edges in G are our linked partition candidates — but we have a few complications to consider:

1. Every partition must be synchronous with every other linked partition; thus we may only consider partitions linked if they belong to the same *clique* in G.

2. Suppose D is our partition dependency digraph. We may combine two partitions only if contracting the related vertices in D does not make D cyclic. A cycle in D would prevent us from compiling a program that invokes kernels in a topological order, which in turn means that we would have to unnecessarily transfer data between threads at run-time.

When determining the set of linked partitions, our goal is find to an optimal set that adheres to those conditions. With the assumption that every vertex in G has an implicit self-loop, we compute an *exact cover* of G where each subset in the collection represents a clique in G if and only if that clique admits stipulation 2. For simplicity, we choose a cover with the fewest number of clique elements. This could be improved by incorporating a cost

function but (as of now) such a function would be asymptotically equivalent to generating a schedule for every option. We leave exploration of this idea to future work.

Because a simulation is an inexact solution, the remaining question is how likely is our set of linked partitions to be correct? Suppose we have two partitions, each with a single `Bounded` input rate. If the probability that these two ports require the same number of items is ϕ, it is obvious that after n simulation iterations the likelihood of a false positive is ϕ^n. Thus assuming we have uniform $[0, 1]$ distribution, the probability we correctly deduce whether these partitions are linked is $1 - 2^{-n}$ so for any $n > 16$ we have $> 99.999\%$ confidence that the assessment is correct. As of the writing of this dissertation, simulations of our most complex programs can iterate at least 100,000 times within 10 seconds.

4.2.4 Final partitioning pass

1. Like with our initial partitioning phase from §4.2.2, we begin with the internal graph G consisting of sets of kernel and streamset nodes connected by channel edges annotated with I/O processing rates and attributes. Each node and edge is associated with an unbounded bitset.

2. Given an arbitrary topological ordering of G, we traverse G's nodes and initialize the bitset of each vertex v to be the disjunction of the bitsets of v's incoming channels.

3. If v is a kernel node with an in-degree of 0, its bitset is flagged with a new marker (that has not been previously associated with any node) because we cannot conservatively assume that the relative output lengths of two arbitrary sources will be equivalent. Similarly, the group of output channels of kernels with an early termination attribute (i.e., `CanTerminateEarly`, `MustExplicitlyTerminate` and `MayFatallyTerminate`) is also assigned an new bit marker. Additionally two new markers are allocated to kernels with `InternallySynchronized` or `IsolatedOnHybridThread` attributes, one for the kernel and the other for the group of outputs, to ensure that such kernels are assigned into their own partition.

4. Every kernel is assigned marker corresponding to its linked partition id, as deduced by the simulator in §4.2.3. The initial partitions within the same set of linked partition are ones that were deduced to have the same symbolic scaling factor for their segment lengths.

5. Like with the initial phase in §4.2.2, after completing the traversal, any kernel node in G with an identical bitset is assigned to the same synchronous partition but then divide each partition into its set of disconnected components. When scheduling, we reason about each partition as a group but such components could benefit from finer grained analysis to minimize the expected memory usage and lifetime. We continue to call these partition components as **linked partitions** and later use this information as a scheduling optimization subgoal to re-fuse components when possible.

4.3 Schedule generation of partitioned program

Schedule generation is a key responsibility of the `PipelineCompiler`. A central hypothesis of this dissertation is that by minimizing memory requirements and selecting a memory layout that facilitates strided prefetchers, we ought to improve cache usage, thereby increasing performance. To reiterate: memory-usage minimization is not the goal of this process; it is a proxy for finding a schedule that minimizes the potential for premature L2-cache eviction.

Although the memory layout is independent of the scheduling process, some schedules may have a smaller memory footprint than others. Our approach is inspired by the hierarchical component design of Ptolemy II [38]. Each partition is scheduled independently and the set of ideal orderings for each partition are determined for them. Based on the ideal intra-partition orderings, an inter-partition graph is generated to denote only the relationships between the inter-partition streamsets and their producer/consumer(s). All other kernel and streamset nodes are removed via edge-contraction. We use this hierarchical design for two reasons: ① interleaving kernel invocations belonging to separate partitions will extend the lifetime of intra-partition streamsets, increasing the likelihood that resulting schedule will not be a minimal memory one. ② partitions naturally cluster tightly-interconnected groups of kernels whose I/O rates are functions of partition root I/O rates. It is unlikely that performance would be increased by interleaving partitions and any exceptions to this case could be corrected by dividing a partition into two linked (i.e., "stride-aligned") partitions.

It should be noted that the only schedules we consider for both intra and inter-partition phases are **topological orderings** of the internal program graph. This minimizes the potential for inter-thread communication, since any other permutation would force one or more kernels to always consume streamset data produced from by another thread.

4.3.1 Intra-partition schedule generation

Because partitions are SDF subgraphs, we can reason them using any SDF scheduling technique. We combine Zitzler et al.'s evolutionary algorithm (EA) algorithm [120, 121] with Yang et al. offline DSA metric [117, 118] to generate and evaluate potential schedules. These are discussed in §3.1.2 and §3.2.3, respectively. To do so, we must transform the internal graph G into something more amenable to both. We discuss how in §3.2.2 but in summary, we convert G into a kernel dependency graph G_D whose edges indicate an ordering constraint due to some incident streamset between two intra-partition kernels. Given a schedule Φ, we compute a streamset interval graph G_I by tracking the use-def relationships in G.

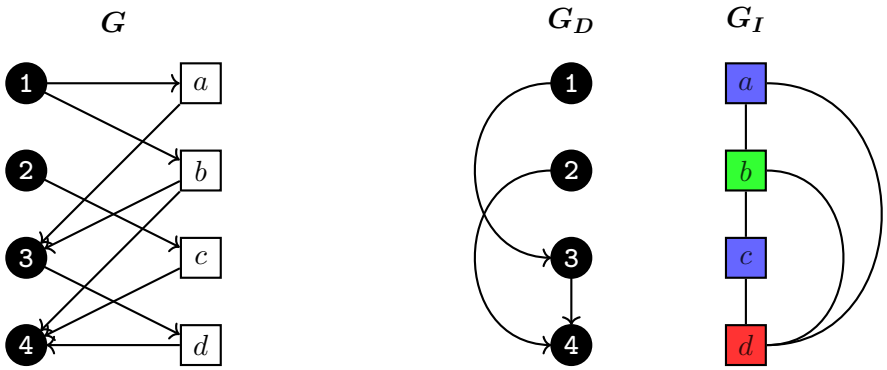


Figure 4.2: Example of a simplified Parabix dataflow dependency and interval graph

Scheduling the dependency graph G_D using any dataflow scheduling algorithm will return an admissible schedule for G. Like Zitzler et al.'s algorithm, described in §3.1.2, every node in G_D represents a kernel invocation and each EA candidate is a topological ordering of the vertices in G_D. The repair function transforms an arbitrary permutation of the vertices into the topological ordering seeded by that permutation. The initial EA candidates are obtained from random topological orderings. Unlike Zitzler et al.'s algorithm, our EA fitness objective is to minimize $\chi(G_I, \mathcal{W}_\mathcal{S})$, where χ is the chromatic number of G_I given the interval weights $\mathcal{W}_\mathcal{S}$, which indicate the required capacity of the streamsets. $\chi(G_I, \mathcal{W}_\mathcal{S})$ is obtained using an adaption of Yang et al.'s weighted interval colouring algorithm. We discuss their algorithm in §3.2.3 and our modifications to it in Appx. A.

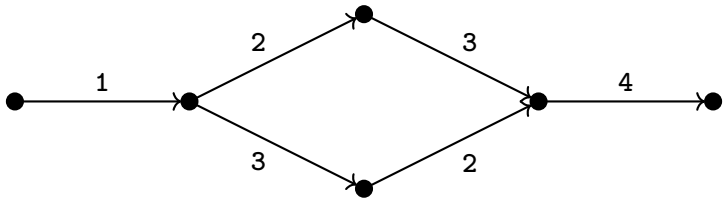


Figure 4.3: Example DAWG for all traversals of Figure 4.3

The EA will naturally converge on a set of solutions that minimize the memory requirement of the partition. These solutions are stored in a minimal directed acyclic word graph (DAWG), constructed via post-order minimization [31] of the of the sequences of kernel id labels corresponding with one of the optimal topological orderings through G_D. Later, we use these graphs to reconstruct an optimal schedule ordering without having to reason about every kernel in the inter-partition schedule generation phase. By definition, each DAWG has precisely one source and sink and the labels along every path has a unique bijective relationship with the kernels within the represented partition.

4.3.2 Inter-partition schedule generation

To understand the objective of our inter-partition schedule, we must first introduce the concept of **partition jumping**. The linear-pipeline generated by the `PipelineCompiler` is an example of static scheduling. For purely fixed-rate (synchronous) programs, static scheduling provides the best performance [71] but a perfect "zero-cost" dynamic scheduler would be better for variable-rate programs. Such a scheduler would selectively enables a kernel at the moment it is fireable and always have enough threads available to invoke it without over-provisioning the system. A linear-pipeline cannot freely invoke kernels but we can proximate the data-driven variation of this behaviour in acyclic (or trivially cyclic) programs. To do so, we make a critical assumption that once a program is in its steady-state that upon invoking kernel $\mathcal{K}$, it will exhaust some incoming channel and that any unconsumed tokens in other channels remain strictly because that channel was exhausted. Obviously, if $\mathcal{K}$ is not invoked, it will produce no output and neither can any of its descendants unless they have an ancestor $\mathcal{A}$ that is not dominated by $\mathcal{K}$ since those descendants may have been blocked previously waiting for $\mathcal{A}$'s output. Stated formally, a potential **jump target** of a $\mathcal{K}$ is a descendant $\mathcal{D}$ in which $\mathcal{K} \notin \texttt{Dom}(\mathcal{D})$.

Definition 5 (Dominator). Let G be a DAG with a solitary source node s. A vertex u dominates v if u is traversed on every $s \rightsquigarrow v$ path; I.e., the dominators of a vertex v are:

$$\texttt{Dom}(v) = \{v\} \cup \left(\bigcap_{u \in \texttt{Pred}(v)} \texttt{Dom}(u) \right)$$

Because partitions are synchronous components, we can extend this concept to the partitions themselves. Specifically, we can construct a graph H from the internal graph G by contracting all kernels and intra-partition streamsets of each partition into a single vertex. If any inter-partition path exists in G, it will be reflected in H. Note that not every channel in H is necessarily variable-rate: attributes such as `LookAhead` or `CanTerminateEarly` may also divide a partition but they do not impede the steady-state data flow. Introducing unnecessary "jumps" only increases the complexity of the compiled `Pipeline`. Even if the "jump handling" code is relegated to the cold path of the program, the Φ-nodes created to merge the incoming processed/produced I/O item counts coming from the various target paths complicate the LLVM optimization passes. Efficiently leveraging partition jumping requires considering the data flow.

Our goal is first to maximize the utility of partition jumping then to minimize the memory cost. Generating an inter-partition schedule is a two-step process. Each step uses a **permutation-based EA algorithm**, similar to what was described in §4.3.1. We describe each phase below:

1) Prioritizing partition jumping

Our goal is to use an EA to converge upon a schedule solution that leverages partition jumping without overly complicating the pipeline code. In Figure 4.4a, we present a simplified inter-partition fork-join graph whose streamset nodes were edge-contracted. The solid and wavy lines represent fixed and variable rate channels, respectively. (b) and (c) are two possible schedules for (a). Here the dashed black lines represent the scheduled execution path of the program and the red lines, the jump targets. Note here that source of a jump target is always a consumer of variable-rate data. At JIT-execution time, there are two places in which we can test whether the data in a streamset is insufficient: at the point of production or consumption. Any conclusions made at production will assume information about the consumer's state and could require retaining additional state in live memory. Consequently, the decision to bypass a partition occurs at the point of consumption by the partition root.

So of (b) and (c), which schedule is preferable? Without accessing extra state information, schedule (b) only permits the program to bypass the immediate partition(s) but (c) provides the ability to skip multiple partitions by way of a partition jump. Consequently, (c) is expected to require fewer I/O tests than (b). Finally, (d) identifies a minor complication: multiple variable rates may exist on a path but the the jump target t of any partition node p ought to be the final node on the path. We generalize this concept by requiring that t is either not dominated by p or is a sink.

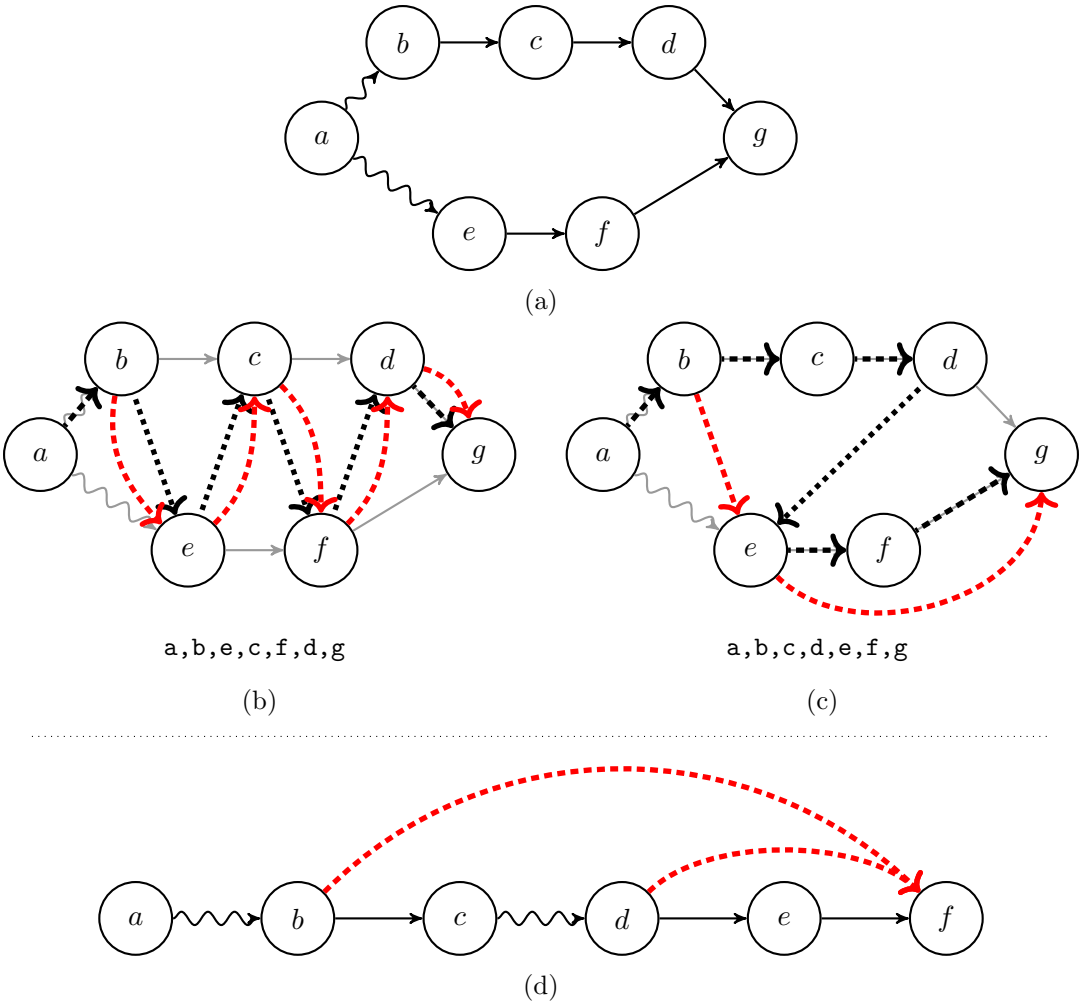


Figure 4.4: Impact of schedules on partition jumping opportunities when a produces variable-rate data.

Using a process similar to the partitioning algorithm (§4.2) we begin this phase by constructing a graph H of the inter-partition dependencies. Associated with every node of H is a bitset. Traversing H in a topological order, we initialize each node to be the intersection of its predecessors' bitset values and mark every source node as well as any streamset node produced at a variable rate with a new bit marker. This provides us with a graph whose nodes are marked by their dominating output rates. To ensure that programs with multiple input sources are properly considered, whenever a bitset is reduced to the empty set after intersection, a new bit marker is added to the set; this guarantees whenever H has an acyclic $K_{2,2}$ graph minor, the program does not wrongly consider divergent subgraphs are originating from a common source. For reasons that will be obvious shortly, we insert a $(t, \mathcal{P}_{out})$ edge into H for every sink t in H except for $\mathcal{P}_{out}$ itself. $\mathcal{P}_{out}$ is the only vertex in H with an empty bitset.

When evaluating the fitness of a candidate schedule $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$, we conceptually generate a multi-digraph J with edges $(\mathcal{P}_i, \mathcal{P}_{i+1})$ and $(\mathcal{P}_i, \mathcal{P}_j)$ when $\mathcal{P}_j$ is its jump target. The $(\mathcal{P}_i, \mathcal{P}_{i+1})$ edges depict the program invocation path when every partition has a sufficient amount of information. The **jump target** $\mathcal{P}_j$ is the first node following $\mathcal{P}_i$ whose bitset is not a subset of $\mathcal{P}_i$ if and only if the bitset of $\mathcal{P}_{i-1} \neq \mathcal{P}_i$. This is the first partition in the linear program whose input dataflow is not strictly dependent on the output of $\mathcal{P}_i$. We refer to J as the **jump graph**. Note, to ensure we generate a linear-pipeline program, J must be acyclic. By Thm. 6, we know J must always satisfy this requirement.

The fitness goal is to find a schedule that minimizes the sum of all $\mathcal{P}_{in} \rightsquigarrow \mathcal{P}_{out}$ path costs in J, where $\mathcal{P}_{in}$ and $\mathcal{P}_{out}$ are partition vertices that contain the implicit pipeline input and output nodes as described in §4.1. At compilation, successive *linked partitions* will be fused into a single partition; thus the cost of traversing any $(\mathcal{P}_i, \mathcal{P}_j)$ edge is 0 if and only if $\mathcal{P}_i$ and $\mathcal{P}_j$ refer to linked partitions and $\mathcal{P}_j$ is not the jump target, and 1 otherwise. The intuition behind this fitness metric is that we want to generate schedules like Figure 4.4c and avoid those like Figure 4.4b. Since the jump targets are a property of the processing rates in H, the sum of the path costs will naturally optimize towards an ordering with the fewest and/or longest jump distances. This tends to result in a depth-first topological ordering of J within regions of a program where jumping is possible but is not limited to them in others. Because $J = (V, E)$ is acyclic and the out-degree of any vertex in V is at most 2, even a naive depth-first enumeration of the paths in J is bounded to $\mathcal{O}(V^2)$.

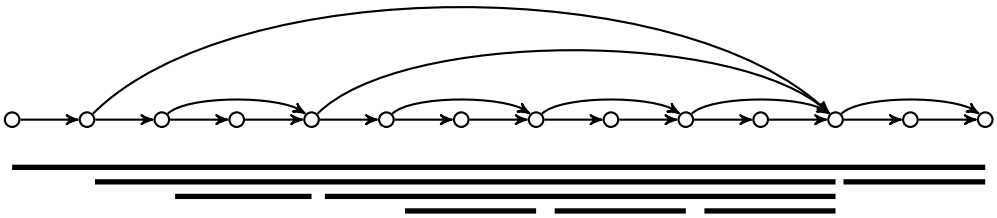


Figure 4.5: Example of jump graph represented as sequence of nested intervals.

Theorem 6. *Jump graphs can be represented as sequences of nested integer intervals.*

Proof. Let a $I_i = [a_i, b_i]$ be an interval and $I_j = [a_j, b_j]$ be a subinterval of I_i. By definition, $a_i \leq a_j \leq b_j \leq b_i$. Because each partition can have at most one jump target, there exists a topological ordering of J s.t. $a_i < a_j$ for all intervals. A natural property of the subset complement relationship each originating node a has with their jump target b is that no node within the $a \rightsquigarrow b$ span can have a jump target after b unless the node is b itself. Consequently, for every subinterval $b_j \leq b_i$. □

2) Optimizing expected inter-partition streamset memory usage

From step 1, we obtain a set of partition orderings $\mathcal{S}$ but as §4.3.1 discusses the partitions themselves may have multiple minimal-memory intra-partition orderings. Even though

inter-partition streamset buffers cannot be precisely reasoned about statically, finding global orderings that minimizes expected memory usage may improve cache utilization.

Every $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n\} \in \mathcal{S}$ is a sequence of partition nodes for which each $\mathcal{P}_i$ maps to one of the DAWGs computed in §4.3.1. Any source-to-sink path through the DAWG refers to one of the optimal orderings through $\mathcal{P}_i$. An optimal program schedule will follow one of the sequences in $\mathcal{S}$, selecting one of the paths through the DAWG at each step. We can use this knowledge to create our initial EA-candidates and construct a candidate repair function to ensure that when given a candidate permutation, we obtain a permissible program ordering.

1. The first step of this phase is to construct a DAWG H from $\mathcal{S}$, substituting the DAWG for each partition node of each sequence in $\mathcal{S}$. However, we do not necessarily need to reason about every kernel in our program. Since we are only concerned with the inter-partition streamsets, we only need to consider the *frontier kernels* that produce or consume such a streamset. For each partition DAWG, we insert a filtered DAWG into H by only inserting the transitions referring to a frontier kernel and conceptually edge-contracting every other transition. We later use a post-processing step to recover an accepting path in each partition DAWG from the chosen path through H.

2. Given H, we compute our initial random candidates by choosing a random path through H and accumulating the transition labels.

3. Building a good EA-repair function that transforms a permutation into a ordering equal to one of the label paths through H is non-trivial. Unlike the topological ordering of §4.3.1 and step 1, seeding our candidate with a permutation sequence may result in a poor exploration of the candidate search space since it will prioritize the early transition choices. If we treat the permutation as a circular list with an randomly-chosen head element, we risk having the resulting candidate being vastly different than the permutation; this would increase the exploration quality of the EA at the expense of its convergence rate. For this problem, we use a **closest-match repair** function, which we detail in Appx. B. It is an ACO-based algorithm that minimizes the Kendall tau distance between the input permutation and output candidate but always generates permissible orderings of H.

4. The fitness function used for this phase is identical to that of §4.3.1. However, the kernel and streamset graph used by our adaptation of Yang et al.'s offline DSA algorithm is limited to the inter-partition streamsets and frontier kernels.

4.4 Thread-local memory layout

Thread-local memory layout is the final analysis phase of the `Pipeline` compilation process. As discussed in §4.2, thread-local buffers are restricted to intra-partition streamsets since

by definition, any data within such buffers is completely produced and consumed when invoking a partition. When assigning a layout, we determine how much thread-local memory required by the program and assign regions of it to the appropriate streamsets. Note that memory layout determination is *independent of schedule generation*. To better understand our memory layout objective, we briefly explain hardware and software prefetching before describing our memory layout algorithm.

4.4.1 Brief review of cache prefetching

Prefetching is a technique for fetching instructions or data into the cache before they are explicitly required by the processor, ideally increasing program IPC by reducing memory subsystem delays. With *software prefetching*, instructions are explicitly added that prefetch arbitrary memory blocks either by the programmer or a compiler optimization pass. *Hardware prefetching*, on the other hand, predicts and proactively prefetches blocks based on the observed data access patterns. In this thesis, we target modern Intel processors. To the best of our knowledge, *recent Intel hardware only supports software and strided hardware prefetching* and places some significant constraints on the latter [58]. We briefly describe strided prefetchers in Def. 7 and direct the interested reader to Bakhshalipour et al.'s survey for a more comprehensive description [7].

Definition 7 (Strided prefetcher). when the processor observes a consecutive sequence of memory-block accesses differing by a constant factor (e.g., $\{A, A+k, A+2k, \ldots, A+(n-1)k\}$) it will prefetch the subsequent $A + nk^{\texttt{th}}$ block.

Intel processors have two types of L1 data (L1D) prefetchers: the data cache unit (DCU) and instruction pointer (IP)-based stride prefetcher. The DCU prefetcher by increasing sequential accesses to recent data. The IP prefetcher associates each access with the requesting instruction's address. Unfortunately, one of the requirements for either is that the data and triggering load instruction must be within the same 4-KB page and the fetch history is invalidated upon any store to the page [58]. Consequently, we should expect L1D prefetchers to only assist in fetching read-only program data and focus on the L2 prefetcher.

Two L2 prefetchers exist: spatial and streamer. However, the spatial prefetcher here is not a correlation-based one; instead when fetching anything into the L2 cache, it attempts to bring in its 128-byte aligned paired line as well. Note, the unrequested line is not automatically brought into the L1D cache.

The **streamer prefetcher** is the primary L2 prefetcher. It monitors L1D requests to predict strided access patterns. To trigger the prefetcher, four cache misses of some k-gap width are required [102]. The first three misses insert a pattern entry into the stream table and the fourth starts the prefetching process. When a pattern is detected, it will fetch the anticipated memory blocks automatically. Intel states that prefetchers will not prefetch across page boundaries and that an explicit load is required for each new page [58].

Empirically, this has been confirmed with one exception: when huge-pages are disabled, the streamer may prefetch up to the first two cache lines of the subsequent page [102].

The 2021 Intel optimization guide states that up to 32 streams can be monitored per-core but only one increasing and one decreasing pattern is permitted for each 4-KB page [58] but a 2020 evaluation suggests no more than 16 pages can be tracked simultaneously [102]. Moreover, a 2020 study found the L2 prefetcher does not appear to be effective (w.r.t., improving memory throughput) for stride lengths (k) of more than 512 bytes in Broadwell micro-architectures [82] but it is unknown whether this has improved in more recent models. Conservatively, however, it appears we can maximize prefetching utility by viewing the streamer prefetcher as a vector-width-aligned sequential prefetcher — a natural use-case for our framework — and align each streamset to 4-KB regions to take advantage of the per-page monitors. As a final note, modern Intel micro-architectures now treat the LLC as a victim cache for the L2 cache [58]; consequently there are no LLC prefetchers.

4.4.2 Prefetch-friendly memory assignment

Based on the information presented in §4.4.1, we know each 4-KB page has a one forward prefetcher monitor and buffers should be 128-byte aligned. Thus we have two minimization goals for the layout: 1) the number of predictor conflicts (i.e., number of streamsets associated with a single kernel that occupy the same page) and 2) the total allocation space.

Given a schedule obtained by §4.3, we process *each partition individually*. For each chosen partition schedule, we apply an approach similar to that of §3.2.2, combining weighted interval graph colouring (§3.2.1) with a permutation-based EA algorithm. Our interval graph I consists of nodes representing streamsets and edges defining overlapping uses in the use-def graph. A key difference is we also annotate I to indicate whether a streamset is an input or output of any common kernel to guide our prefetcher-aware fitness metric.

Each candidate is a permutation of the streamset nodes. The fitness function uses this permutation as a vertex list to feed into a first-fit colouring algorithm to calculate its memory requirements. To incorporate prefetcher awareness, we modify the standard greedy approach by incorporating a function $\textsc{Alignment}(x, y)$ that returns 4096 when streamsets x and y are used by a common kernel and 128 otherwise. The modification is formally described in Def. 8. The EA algorithm optimizes towards finding a minimal-memory candidate.

Definition 8 (Modified first-fit colouring algorithm). Let C_v be the hues assigned to vertex v. A first-fit colouring accepts a vertex list $\mathcal{L} = \{v_1, v_2, \ldots, v_n\}$ and sequentially assigns each $v_i \in \mathcal{L}$ the colours $C_{v_i} = [a \lfloor k/a \rfloor, a \lceil (k+\mathcal{W}(v_i))/a \rceil)$ s.t. k is the smallest positive integer in which for every adjacent vertex $v_j \prec v_i$, $C_{v_i} \cap C_{v_j} = \emptyset$ and $a = \textsc{Alignment}(v, v_i)$.

4.5 Hybrid pipeline model

As discussed in §2.1.5, this framework primarily uses a *relaxed fixed-data* pipeline model but has limited support for a hybrid model, as described by Mastoras [78]. With this model, a single thread is rationed to execute a predetermined subset of kernels, which are removed from the schedule on the main fixed-data thread(s). Unlike Mastoras's approach, the framework does not support dynamic scheduling and cannot take advantage of work stealing. Moreover, we cannot easily automate this process in Parabix and instead rely on kernels being annotated with the `IsolatedOnHybridThread` attribute to identify whether to assign them to the hybrid thread. Even with the advances in static performance analysis, the most likely kernel(s) to be isolated would belong to a family of kernels. As described in §2.1.7, kernels with a `Family` attribute could be compiled significantly later the pipeline and performing static analysis on general LLVM functions at every program execution to identify which pipeline variation to utilize would likely exceed the benefit of doing so.

We provide a proof of concept implementation of hybrid pipeline parallelism for the purpose of evaluation in §5.6. Both partitioning and scheduling analysis occur as normal but kernels with the `IsolatedOnHybridThread` attribute are placed into unique partitions. The key difference is in code compilation: two thread functions are generated, one for the fixed-data kernels and another for the hybrid-thread ones. Because the consumed value must be a monotonically non-decreasing value, a pair of thread-specific scalars for consumed item counts for any cross-thread streamsets are added to the shared pipeline state. The streamset producer considers the minimum of both to be actual consumed count. The major complication, however, is cross-thread synchronization. With the standard fixed-data model, kernel invocations occur in a lock-step pattern. Data is typically produced and consumed at consistent rate and buffer expansions tend to be data dependent. With the hybrid model, if we cannot statically bound the size of the cross-thread buffers then we are dependent on the system reaching an production/consumption equilibrium quickly or risk front-loading the program with a large number of buffer expansions. To mitigate this, we bound the synchronization number of both the thread types to be no more than $(\#\text{ of threads} - 1) +$ the synchronization number of the alternate thread type. This helps the programs reach an equilibrium point faster as neither thread-type is permitted to run too far ahead of the other and because invocations scale to their available input, they can catch up whenever this occurs.

Chapter 5

Evaluation

In this chapter, we perform a quantitative study of various Parabix programs to determine their multi-thread speedup relative to their single threaded linear-pipeline cost. We also assess the current framework against earlier versions of the framework, program variations, and a selection of non-Parabix benchmark applications. In our comparison against earlier versions of the framework, we select two key points: a 2017 version that corresponds with Dan Lin's last repository submission and a 2020 version that is the last version not to include some notion of partitioning or partition jumping. *For brevity, we refer to the 2017, 2020 and current versions of the framework as* `2017`*,* `2020` *and* `CURR` *within this chapter.*

5.1 Experimental platform and methodology

Each case study was executed on all three platforms listed in §5.1.4. JIT-compilation was performed using LLVM 12.0, which was the most recent official release of LLVM at the start of data collection for this dissertation and had better `AVX-512` instruction selection than earlier versions. To instrument a Parabix program, we surround the JIT-execution call with PAPI[1] instrumentation. For our evaluations, we limit instrumentation to measurements of the *process thread*, which is responsible for memory allocation and deallocation and supplementary thread creation and teardown as well as the first of the k threads execution. When $k = 1$, synchronization (§2.1.5) is omitted; thus a single-threaded run represents a lower-bound of the total work performed compared to other configurations. Non-Parabix programs are instrumented similarly but because they lacked discrete JIT-compilation and execution phases, we instrumented their main work functions.

Unless specified otherwise, every reported value is based on the geometric mean of 50 trials per configuration. A **configuration** is a combination of a program, platform, data set (or appropriate command-line arguments), segment-length and thread-count. The

[1]The PAPI library [108] is a low-level API for reading of various hardware performance counters in most microprocessors. For additional information, we refer the reader to `https://icl.utk.edu/papi/`.

segment-length is a rough measure of the amount of data each kernel will process per logical invocation and the **thread-count** is the number of threads provisioned to the program at JIT-execution. With each experiment, we vary the *thread-count from 1 to 4* and set the *segment-length to one of 4KB, 16KB or 32KB*. These three settings were chosen because the minimum length that all three frameworks supports is 4KB, 16KB was empirically found to provide the best average performance, and settings above 32KB provide only negligible improvements to performance for sufficiently large files.

To eliminate complications due to hard drive write speed, the output of each trial is piped out to `/dev/null`. To guarantee that the page cache contains the dataset under test, data collection begins with two discarded trials.

In §5.2, we compare the observed multi-threaded program speedup to the theoretically optimal pipeline-parallel speedup. In §5.3, we evaluate the average performance improvement of our selected platforms. In §5.4 and §5.5, we compare `2017` and `2020` to `CURR`, respectively. In §5.6, we compare `icgrep` in `CURR` against a hybrid-thread variant. Finally in §5.7, we contrast `CURR` with other (non-Parabix) programs.

We describe methodology for §5.2, §5.3 and §5.7 in their own sections but due to their similarity, detail the presentation for the remaining studies here. In §5.4 to §5.6, compares two framework variants (e.g., `CURR` and `2017`) used to compile the same program with a particular configuration. For each of three platforms, we compare the geometric means of the recorded PAPI measurements for variants a and b, where b is always the framework variant under consideration. We identify the best performing segment-length when allotted k threads on that platform for variant a but choose the best ranked segment-length for variant b. **Ranking** is determined by computing the median execution time of each each program configuration in a thread-count group and counting the number of times in which a particular segment-length resulted in the lowest median time. Tie breaks were determined similarly, substituting third quartile for median. We chose this strategy because the platform and thread-count are always known prior to JIT-compilation but the effect of other arguments typically cannot be predicted a priori (i.e., until after JIT-execution time.) To summarize the data, these sections (and their accompanying appendices) report the **normalized difference** between variants a and b) of total cycles (**CYC**), L2 and L3 misses (**L2M** and **L3M**) of the main process thread between two variants for the chosen segment-length (**SL**), reported in KB. The normalized difference is simply,

$$\frac{\texttt{measurement}_a - \texttt{measurement}_b}{\texttt{file size in bytes}} \tag{5.1}$$

Positive values for Eq. 5.1 always denote an improvement for b over a. We highlight negative values in red for visual ease. Supplementary tables for every case study can be found in Appx. D but any interesting details will be discussed here.

5.1.1 Applications

This section briefly describes the programs used within the following case studies. Simplified dataflow diagrams for each program are provided in Appx. C.

- `base64`: a binary-to-text encoding scheme that that converts each 3 input bytes (rounded up) into 4 output digits. See RFC 4648 for a complete specification.

- `csv2json`: a Parabix-based CSV to JSON text converter. See RFC 4180 and RFC 7159 for file format specifications.

- `editd`: a `Pablo`-based approximate string matcher that permits up to k insertions, removals or substitutions per match. A full description of the `editd` algorithm is available in Ch 6.3 of Dan Lin's dissertation [75].

- `gb18030`: a GB-10830 to UTF-8 transcoder, predominately written in `Pablo`.

- `icgrep`: a standalone replacement for the `grep` regular expression matcher with complete Unicode level 1 and nearly-complete level 2 support [33]; Like with many `grep` applications, `icgrep` supports colourized output, highlighting matches using terminal colourization codes. This, however, results in a vastly different program structure (shown in Appx. C.5 vs. Appx. C.6). Consequently, for the purpose of these case studies, we consider `icgrep` with and without colourization as being two independent programs.

- `u8u16`: a UTF-8 to UTF-16 transcoder.

- `u32u8`: a UTF-32 to UTF-8 transcoder.

- `ztf-hash`: a preliminary version of a parallel compression and decompression application for valid UTF-8 text.

5.1.2 Datasets

For the following experiments, we have two different data sets. The text datasets listed in Tbl. 5.1 are wikimedia data dumps and are used for the majority of the experiments. Because `csv2json` requires CSV input, for those studies we use the CSV datasets in Tbl. 5.2. These were obtained from `www.stats.govt.nz/large-datasets/csv-files-for-download`. To obtain our test cases for `gb18030` and `u32u8`, we used `iconv -f UTF-8 -t GB18030 -c` and `iconv -f UTF-8 -t UTF-32 -c`, respectively, to convert the files listed in Tbl. 5.1 to the appropriate character set format. `ztf-hash` decompression files were the result of a preliminary `ztf-hash` compression pass. Finally, for the `editd` case studies, we select 100

random reads from the Pacbio dataset[2], truncated to the first 20 base pairs. The reference genome is a $250MB$ sequence downloaded from the UCSC genome browser[3].

Id	File	Size (B)	Chars
Arwiki	arwiki-20211101-pages-articles-multistream3.xml-p1205739p2482315	1,432,876,500	1,041,170,631
Enwiki	enwiki-20210501-pages-articles-multistream-index.txt	992,394,700	989,858,628
Ruwiki	ruwiki-20211101-pages-articles-multistream1.xml-p1p224167	1,765,588,685	989,858,628
Zhwiki	zhwiki-20210501-pages-articles-multistream1.xml-p1p187712	681,187,052	432,454,802

Table 5.1: Primary UTF-8 text datasets

Id	CSV Size (B)	Columns	Rows	JSON Size (B)
Census	857,672,667	7	34,959,673	2,850,373,942
Finance	5,881,081	7	1,658,793	13,032,659
Trade	82,256,023	10	37,081	233,206,026

Table 5.2: CSV datasets

Id	GB-18030	UTF-32	ZTF-Compressed
Arwiki	2,213,808,448	4,164,682,528	1,138,902,030
Enwiki	993,381,295	3,959,434,512	941,924,947
Ruwiki	1,774,653,801	4,544,278,528	1,613,744,666
Zhwiki	557,967,891	1,729,819,208	630,382,273

Table 5.3: Converted text datasets file sizes

5.1.3 RE queries

Seven queries were selected for the `icgrep` case studies: the five REs used in [27], `\p{Greek}` and `\X`. The ones chosen from [27] were based on real life examples. `\p{Greek}` matches any single Greek character and is reflective of a typical `icgrep` workload. `\X` is a pathological case as it matches any `UNICODE` grapheme.

[2] ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR306/SRR306842/SRR306842_1.fastq.gz

[3] http://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/chromFa.tar.gz

Name	Expression
@	`@`
Date	`([0-9][0-9]?)/([0-9][0-9]?)/([0-9][0-9]([0-9][0-9])?)`
Email	`([^@]+)@([^@]+)`
URIOrEmail	`([a-zA-Z][a-zA-Z0-9]*)://([^/]+)(/[^]*)?|([^@]+)@([^@]+)`
Xquote	`["']|&quot;|&apos;|&#0*3[49];|&#x0*2[27];`
\X	`\X`
\p{Greek}	`\p{Greek}`

Table 5.4: REs used in `icgrep` evaluations

5.1.4 Platforms

For our case studies, we assess the performance on three different architectures of varying age and levels of SIMD support. Our primary target platform is the `AVX2` (shown in Tbl. 5.6) but note that systems supporting the `AVX-512` instruction set extensions are becoming more common as of the 10^{th} generation Intel processors. Because `Pablo` is intrinsically linked to SIMD acceleration, and `IRBuilders` are specialized for each of these instruction sets, these platforms give a fair cross-section of our expected execution behaviour. Most information is reported verbatim from `/proc/cpuinfo` but cached and buffered reads were found via `hdparm -Tt`.

Physical cores	4	L1 cache	32/32 KB
HW threads	8	L2 cache	256 KB
Base clock	1.60 Ghz	L3 cache	8 MB
Ubuntu	16.04	Cache-line size	64 bytes
Buffered reads	124.5 MB/s	Cached reads	13.6 GB/s
Hard drive	SATA HDD	Memory	8 GB

Table 5.5: `SSE-4.2`

Physical cores	4	L1 cache	32/32 KB
HW threads	8	L2 cache	256 KB
Base clock	1861 Mhz	L3 cache	8 MB
Ubuntu	19.04	Cache-line size	64 bytes
Buffered reads	701.0 MB/ s	Cached reads	7.5 GB/s
Hard drive	NVME SSD	Memory	16 GB

Table 5.6: `AVX2`

Physical cores	4	L1 cache	32/32 KB
HW threads	4	L2 cache	1 MB
Base clock	1.36 Ghz	L3 cache	8 MB
Ubuntu	18.04	Cache-line size	64 bytes
Buffered reads	1570.0 MB/s	Cached reads	7.3 GB/s
Hard drive	NVME HDD	Memory	8 GB

Table 5.7: `AVX-512`

5.2 Average multi-thread speedup evaluation

In this section, we evaluate the multi-threaded speed up of the programs listed in §5.1.1 using the `CURR` framework. Our goal is to compare the speedup exhibited by these programs to their ideal pipeline-parallel speed up. Programs using a strict pipeline-parallelism are limited to a maximum speed up of ($\texttt{total single-threaded time}/\max(\texttt{total kernel time})$) but as we discussed in §2.1.5, this framework supports data-parallel execution of a subset of statefree kernels. Because most of our programs are dominated by stateful kernels, we still see this metric is still a useful indicator of improvement.

Methodology

Each program execution has a particular configuration, composed of its architecture, thread-count, segment length and any program arguments. For most programs, the only argument is its dataset, selected as appropriate from §5.1.2. For `icgrep` we use the REs listed in Tbl. 5.4 but isolate the output colourization mode, due to the added complexity they add to the pipeline. Finally, for `editd` we combine the results for edit-distance=$1, 2, 4$, and 8. Despite the fact that the complexity of the pattern matching kernels is $\mathcal{O}(d^2)$ w.r.t. the edit distance d, the overall program balance is nearly identical.

Each case study consists of two figures: ① a box plot that indicates the range of observed speed ups relative to the minimum single-threaded cost of any of the segment length for a given configuration. That is, each speed up value is ($\textit{minimum single-threaded cost}/\textit{observed value}$). This hints at what segment length is likely to best improve the expected and peak performance for each application. ② a weighted scatter plot for the best segment length that describes how close the observed speed ups come to their theoretical ideal strict pipeline-parallel cost (Eq. 5.2). We determine the best segment length with the ranking method outlined in §5.1. Each y-value is computed as ($\textit{ideal cost}/\textit{observed value}$) but to highlight the density of these values, we divide the range into bins of 5% intervals. The percentage of the observed values that fall into each bin dictates the size and colour of each point and its position on the plot is determined by geometric mean of its binned values. Each chart depicts the

regression line with a solid black line and dashed line showing the achieved speedup with $(\texttt{total single-threaded time}/\max(\texttt{total kernel time}))$ threads according to the regression model. Note that a perfect linear speedup is one whose regression model is $y = 100\%$ and lower values represent sub-optimal results.

$$\textit{ideal cost} = \frac{\texttt{total single-threaded cycles}}{\min(\frac{\texttt{total single-threaded cycles}}{\max(\texttt{individual kernel cycles})}, \#\ \texttt{of threads})} \tag{5.2}$$

Practically speaking, in a fully stateful application the *ideal cost* is not an achievable one for more than one thread: cross-thread communication and synchronization costs are unavoidable overheads that are ignored in Eq. 5.2. Moreover, the cost of the most-expensive kernel varies depending on the configuration. Appx. D.1, show this cost tends to increase when the thread-count increases but rate in which it grows depends on kernel workload imbalance and the number of kernels in the program; pipelines with a larger number of kernels can better absorb sporadic dataflow perturbations. For the purpose of this study, we view Eq. 5.2 as a coarse over-approximation of optimality for strict pipeline-parallel programs but note that applications with significant amounts of statefree work may exceed the theoretical 100% limit.

General analysis and recommendations

According to our regression models, all but one of the tested applications have an *ideal cost* metric over 80% with an optimal number of thread (w.r.t. Eq. 5.2). The exception is `icgrep colours=always`, which only achieves 71% of the *ideal cost*. `icgrep` is unique compared to the other programs in that the RE greatly affects the running time and data set dictates which branches are taken in the RE kernel and subsequent output generation kernels. Incorporating the RE into the ranking algorithm improves the regression model by $< 0.5\%$. Appx. D.2 hints to the reason. When comparing a typical run of `icgrep`, such as shown in Figure D.1, bursts of activity occur sporadically throughout the program. Although this volatility is not unique to `icgrep` — as evident with the diagrams for the better behaved programs `gb18030` and `u32u8` in Appx. D.2.2, Appx. D.2.3, respectively — in other programs every kernel consistently has work to process whereas some of the kernels in this mode for `icgrep` rapidly shift between having no work and several hundred strides of work in very short intervals. This happens because RE matches tend to be sparse — but sparsity itself is not a problem: when a match occurs on one thread, our synchronization mechanism forces all other threads to wait for it to process the match(es). Partition jumping is intended to alleviate this burden by short-cutting I/O tests but must still obey the synchronization constructs.

Several ways that might avoid this issue are: ① an n-stage execution that divides the dense and sparse portions of a program (i.e., textual analysis and match reporting regions). ② an independent maximum stride bound for each kernel that dynamically increases or

decreases based on recent activity. ③ dynamic scheduling of clusters of partitions that correspond with dense and sparse subdivisions of the program graph. Each of these solutions have potential problems. ① apart from being a programmer responsibility, an n-stage execution requires that all of the intermediary output is stored in memory, which can be problematic in denser than expected cases, such as a pathological any character `icgrep` case. ② adjusting the stride bound at run time could increase data transfers between threads when the expected number of strides is low and could be slow to react to brief bursts depending on the formula used to calculate the segment length(s). ③ dynamic scheduling can mitigate the intermediary output issue but would require a manager to selectively choose when to invoke the clusters, which — if attempting to guarantee optimal throughput — would require awareness of the likelihood that selecting one would result in executing a sufficient amount of useful work.

Interestingly, both `base64` and `u8u16` exceed the maximum pipeline parallel speedup by a significant factor. Between 70% and 90% of work performed by these programs comes from statefree kernels. Even though these kernels must execute in lockstep and no thread can "run ahead" of a thread that is processing a prior segment, there is significant data parallelism evident in these programs. While these experiments should be treated as outliers rather than typical use cases, they do show the importance of good program design.

5.2.1 Case study: `base64`

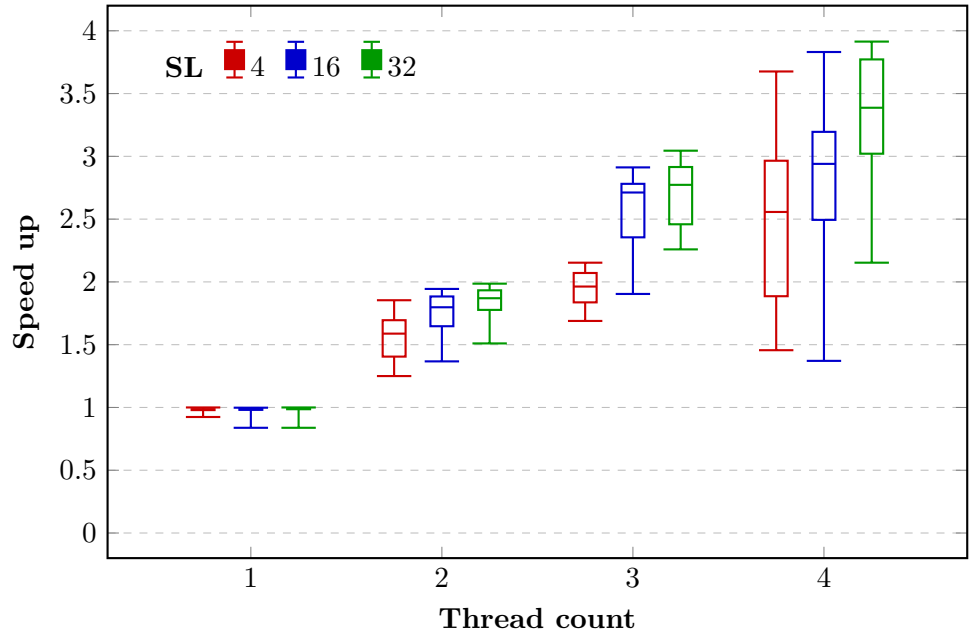


Figure 5.1: `base64` speed up compared to minimum single-thread cost

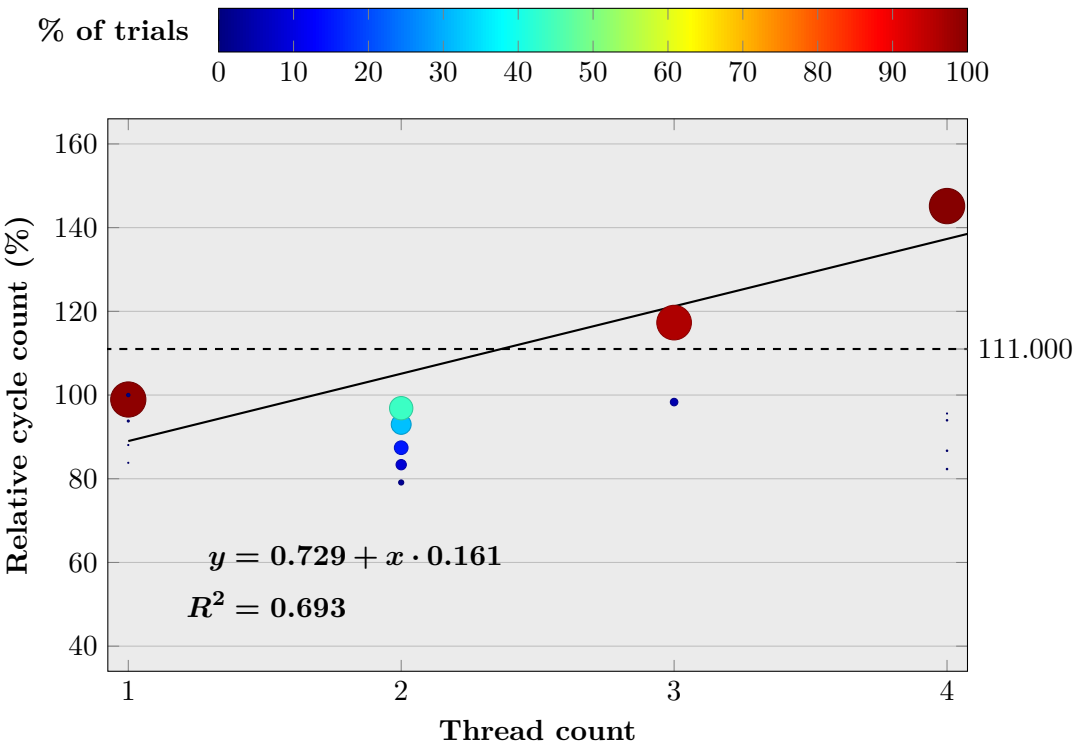


Figure 5.2: Speedup of `base64` compared to ideal pipeline-parallel cost

5.2.2 Case study: `csv2json`

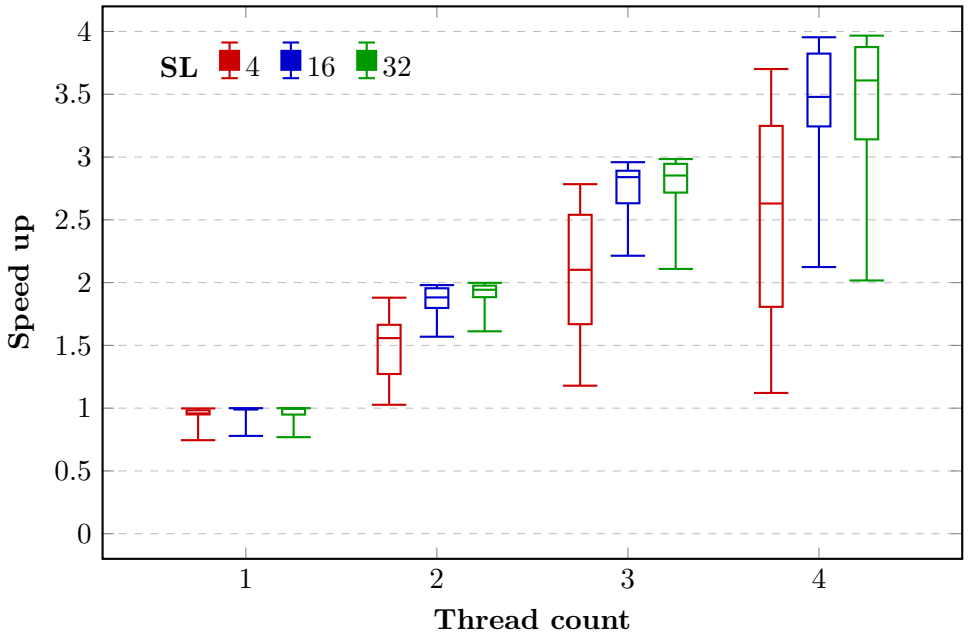


Figure 5.3: `csv2json` speed up compared to minimum single-thread cost

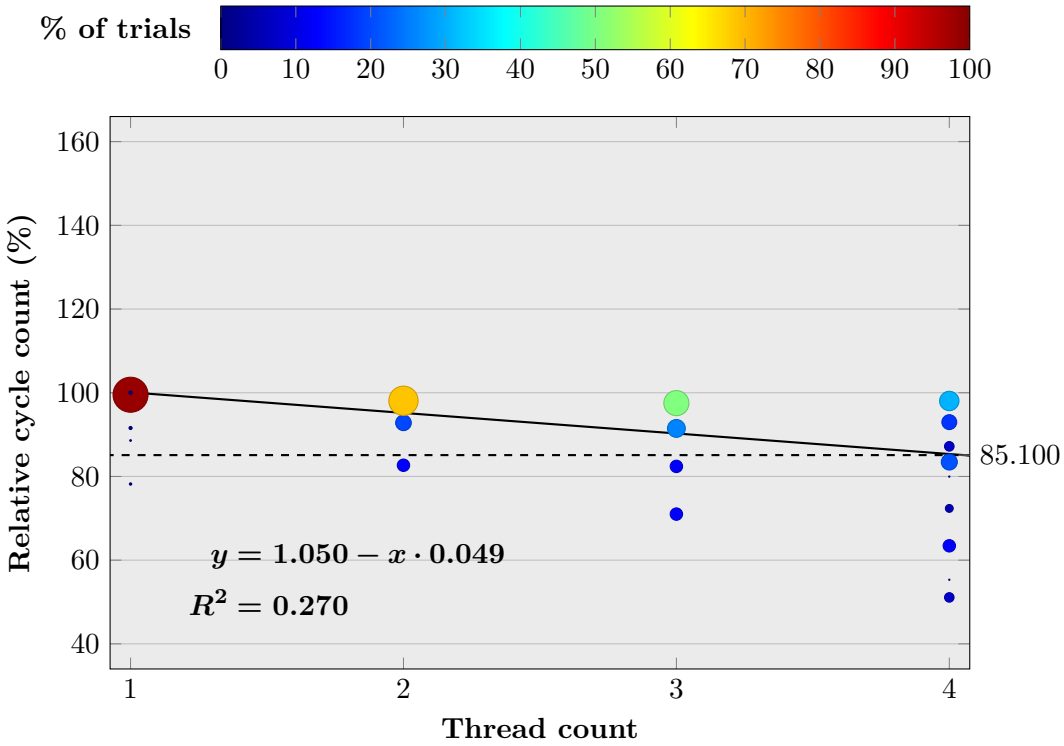


Figure 5.4: Speedup of `csv2json` compared to ideal pipeline-parallel cost

5.2.3 Case study: `editd`

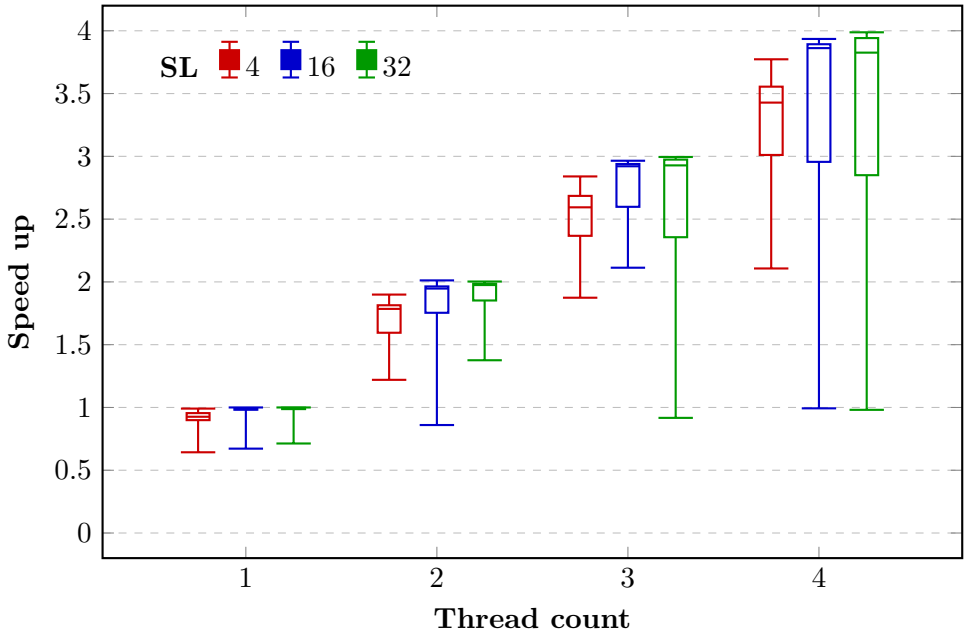


Figure 5.5: `editd` speed up compared to minimum single-thread cost

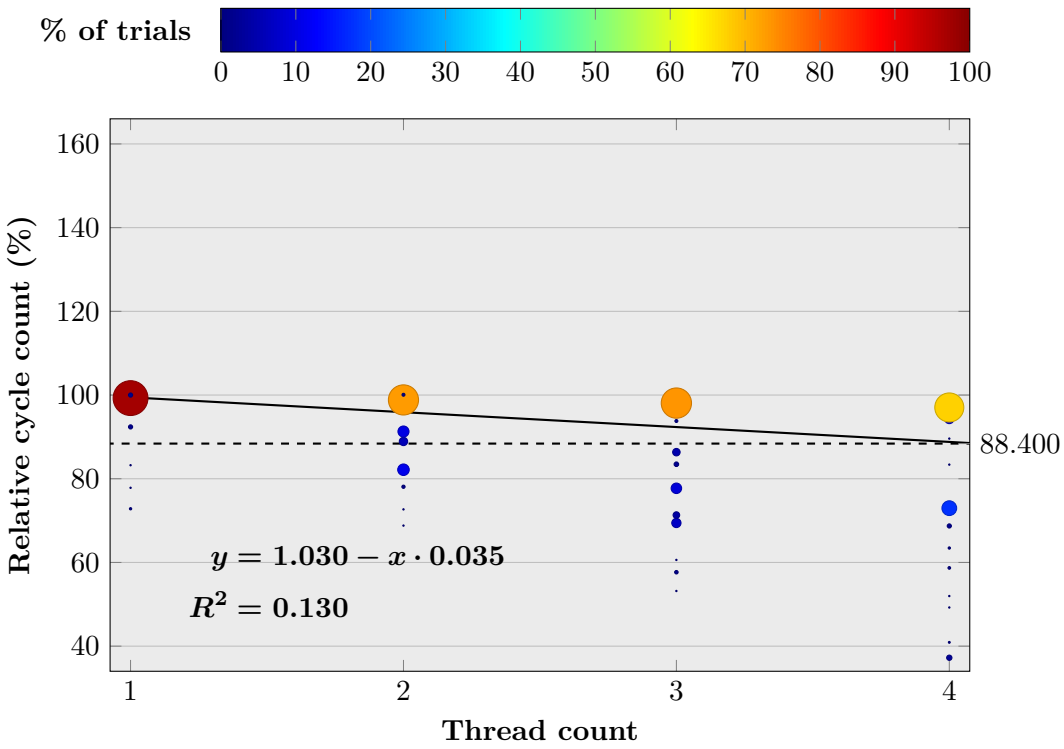


Figure 5.6: Speedup of `editd` compared to ideal pipeline-parallel cost

5.2.4 Case study: `gb18030`

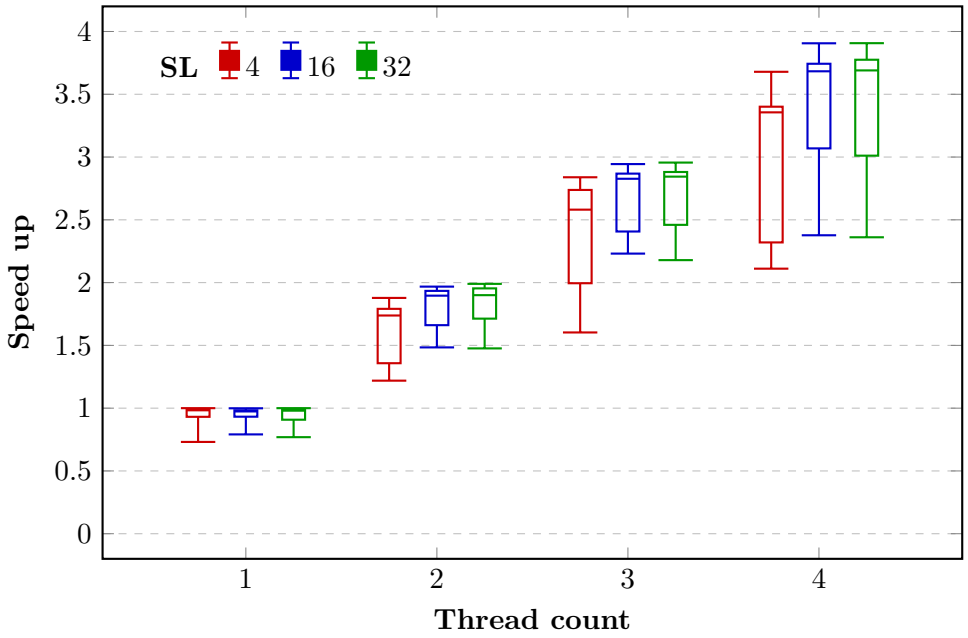


Figure 5.7: `gb18030` speed up compared to minimum single-thread cost

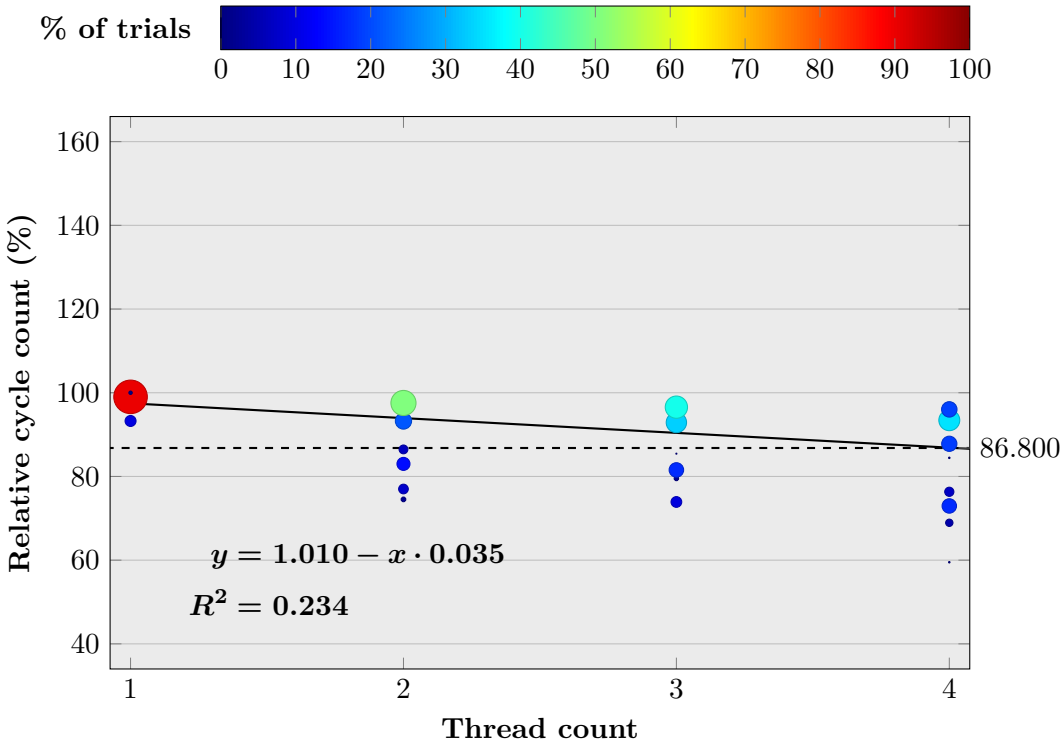


Figure 5.8: Speedup of `gb18030` compared to ideal pipeline-parallel cost

5.2.5 Case study: `icgrep -colours=never`

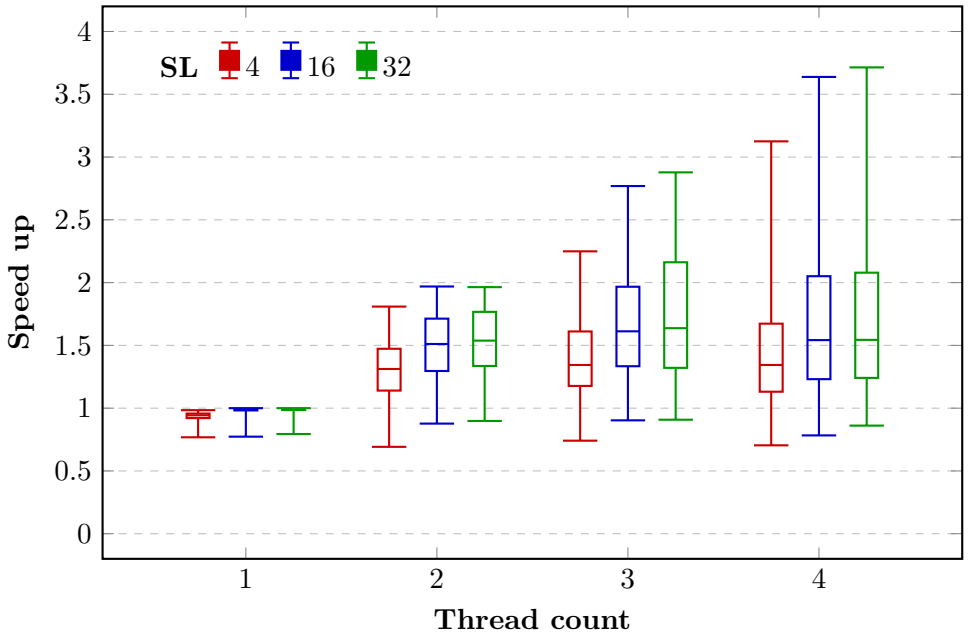


Figure 5.9: `icgrep -colors=never` speed up compared to minimum single-thread cost

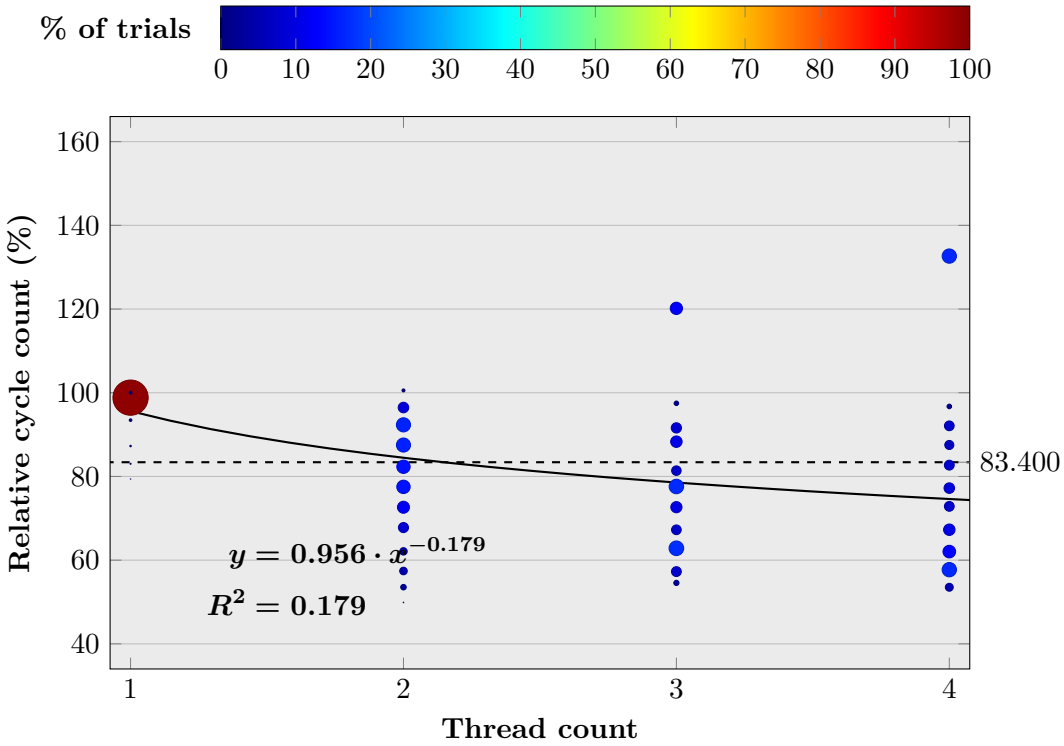


Figure 5.10: Speedup of `icgrep -colors=never` compared to ideal pipeline-parallel cost

5.2.6 Case study: `icgrep -colours=always`

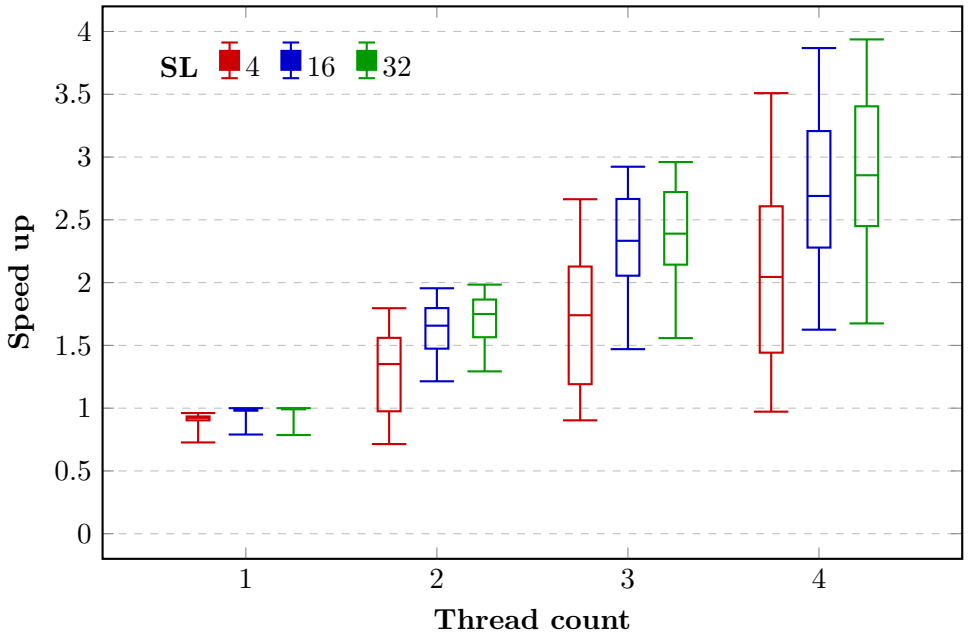


Figure 5.11: `icgrep -colors=always` speed up compared to minimum single-thread cost

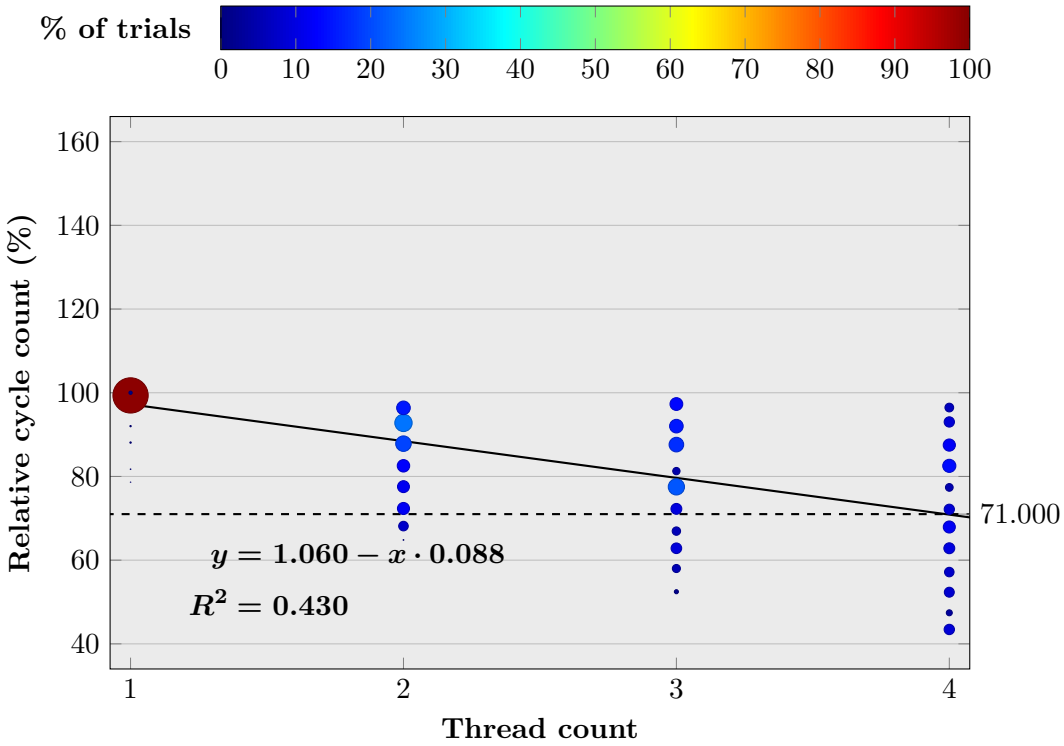


Figure 5.12: Speedup of `icgrep -colors=always` compared to ideal pipeline-parallel cost

5.2.7 Case study: `u8u16`

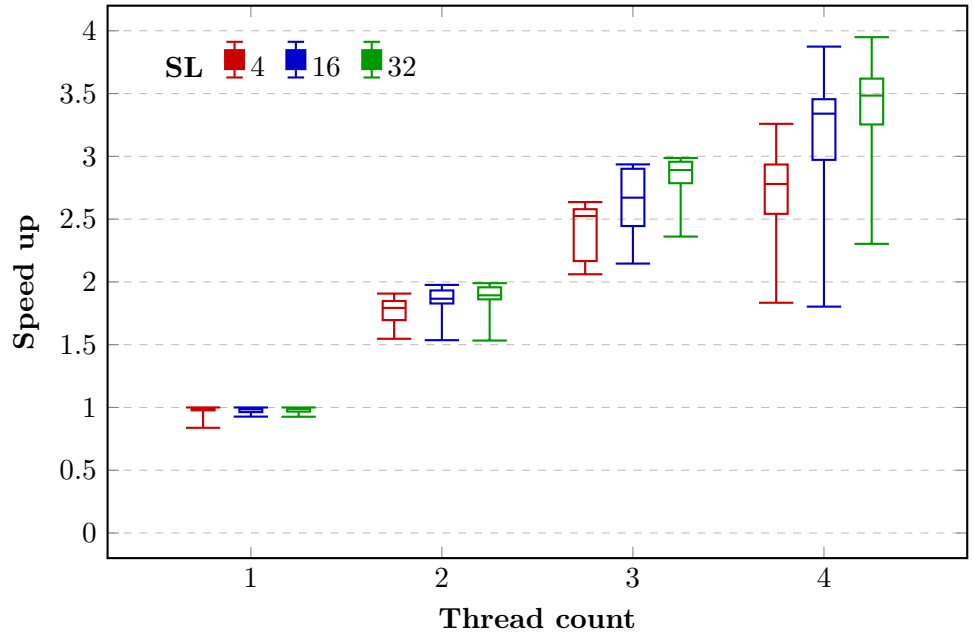


Figure 5.13: `u8u16` speed up compared to minimum single-thread cost

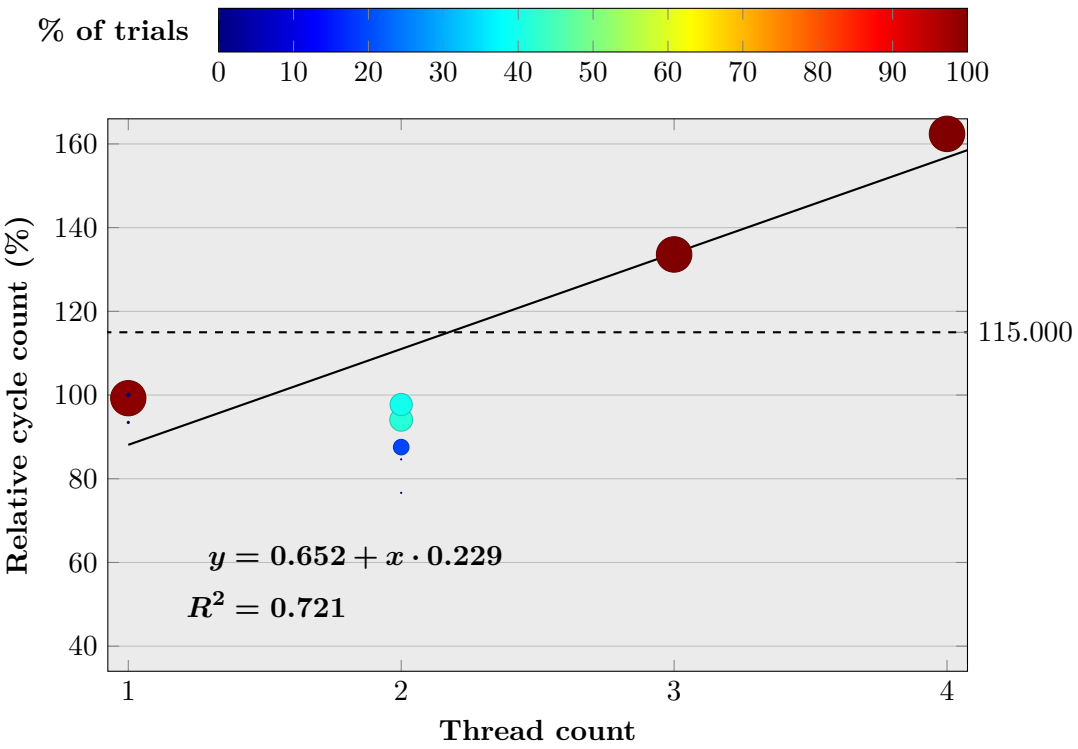


Figure 5.14: Speedup of `u8u16` compared to ideal pipeline-parallel cost

5.2.8 Case study: u32u8

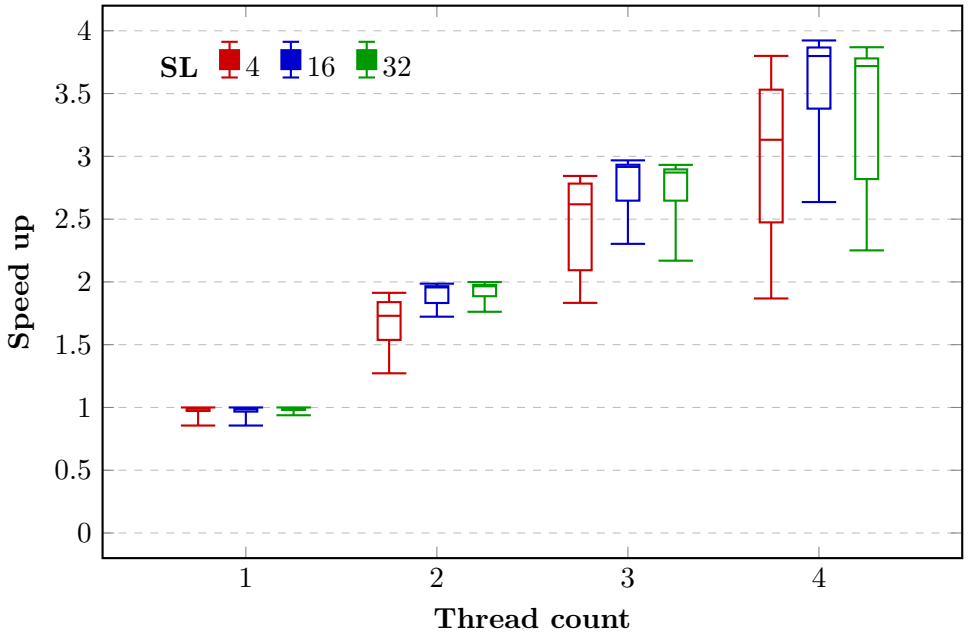


Figure 5.15: u32u8 speed up compared to minimum single-thread cost

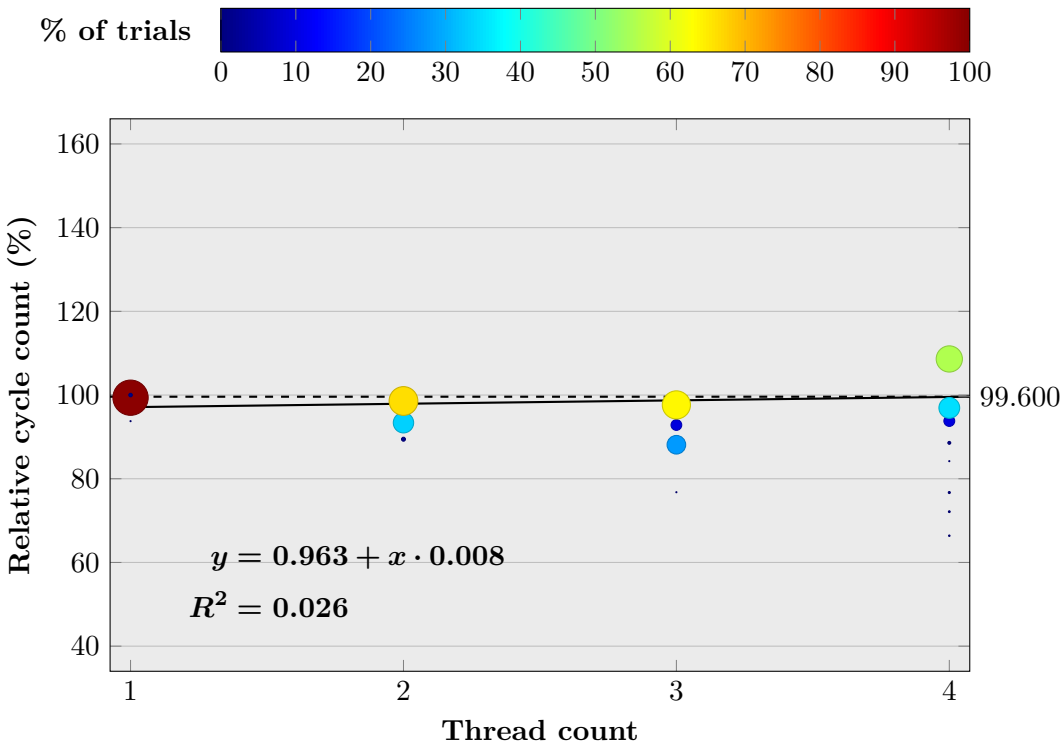


Figure 5.16: Speedup of u32u8 compared to ideal pipeline-parallel cost

5.2.9 Case study: `ztf-hash compression`

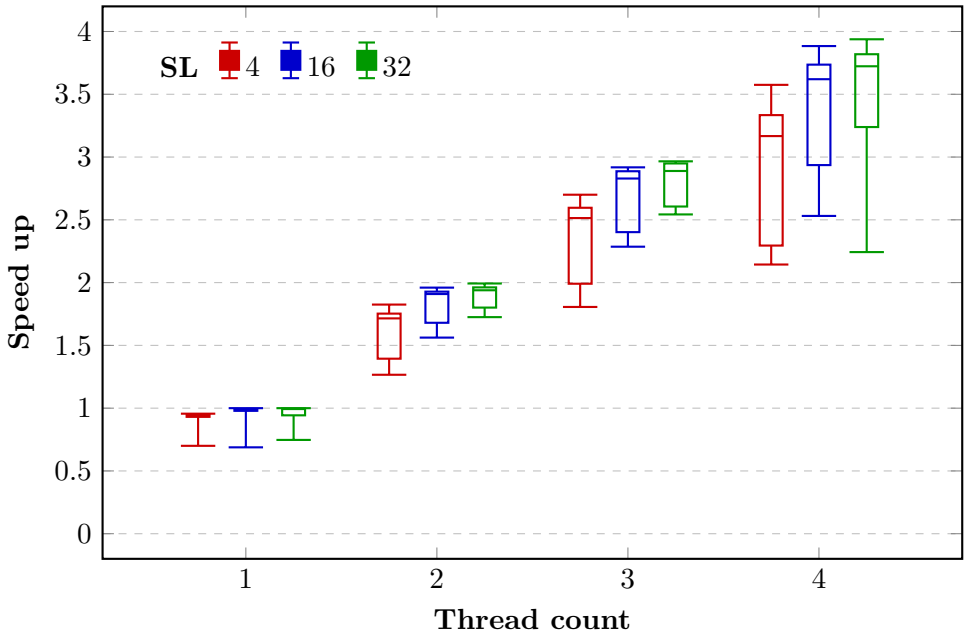


Figure 5.17: `ztf compression` speed up compared to minimum single-thread cost

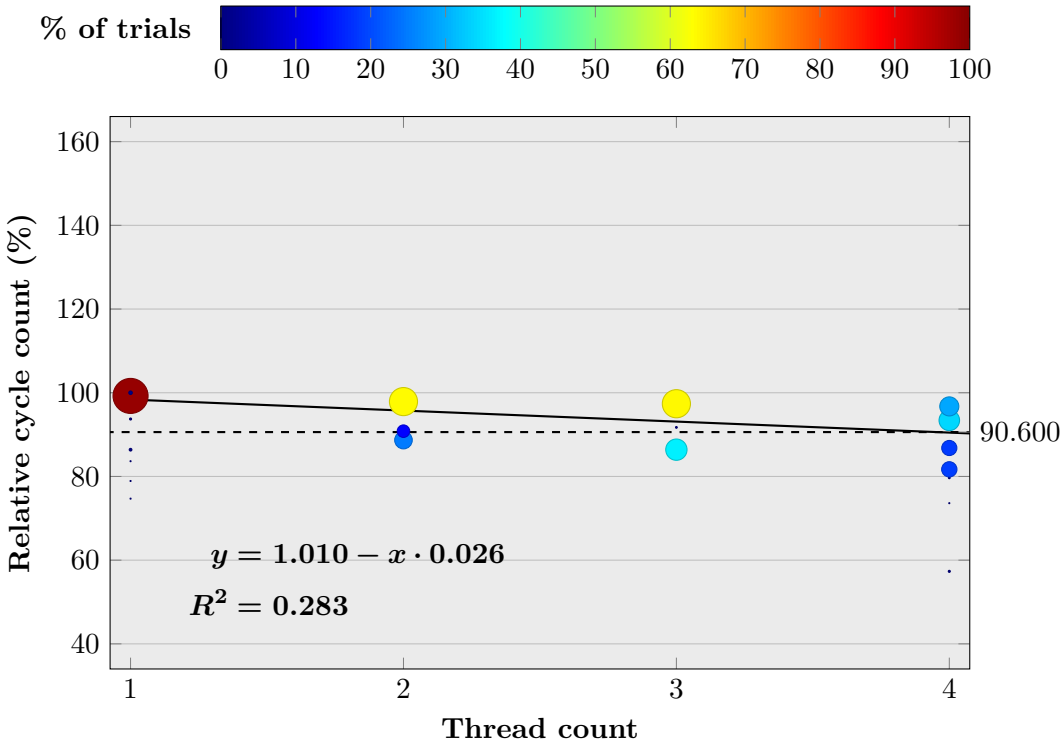


Figure 5.18: Speedup of `ztf compression` compared to ideal pipeline-parallel cost

5.2.10 Case study: `ztf-hash decompression`

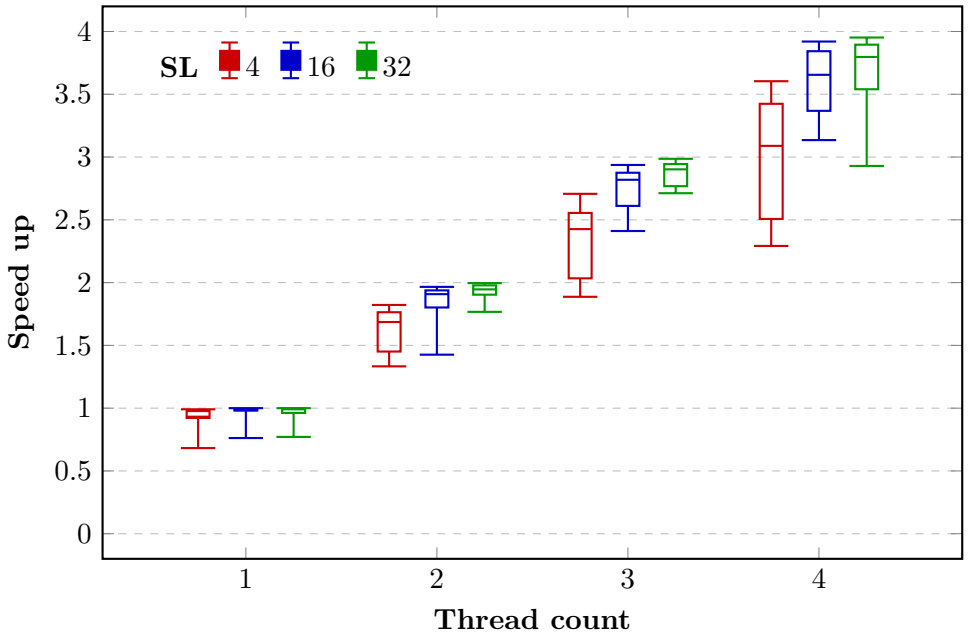


Figure 5.19: `ztf decompression` speed up compared to minimum single-thread cost

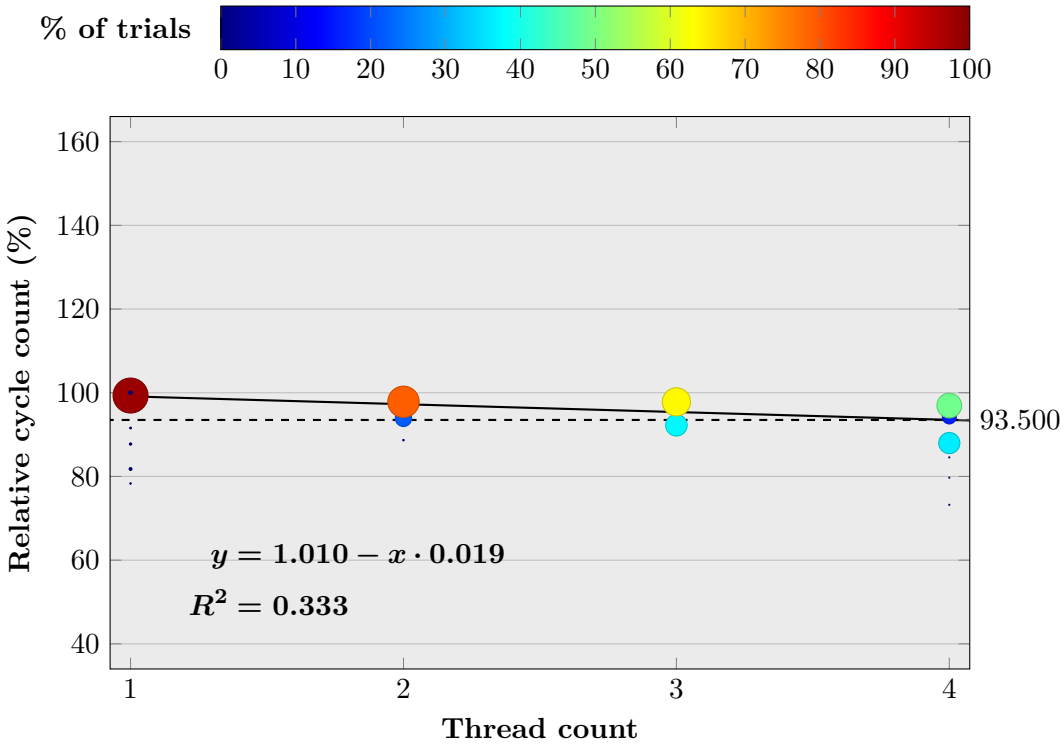


Figure 5.20: Speedup of `ztf decompression` compared to ideal pipeline-parallel cost

5.3 Average platform speedup evaluation

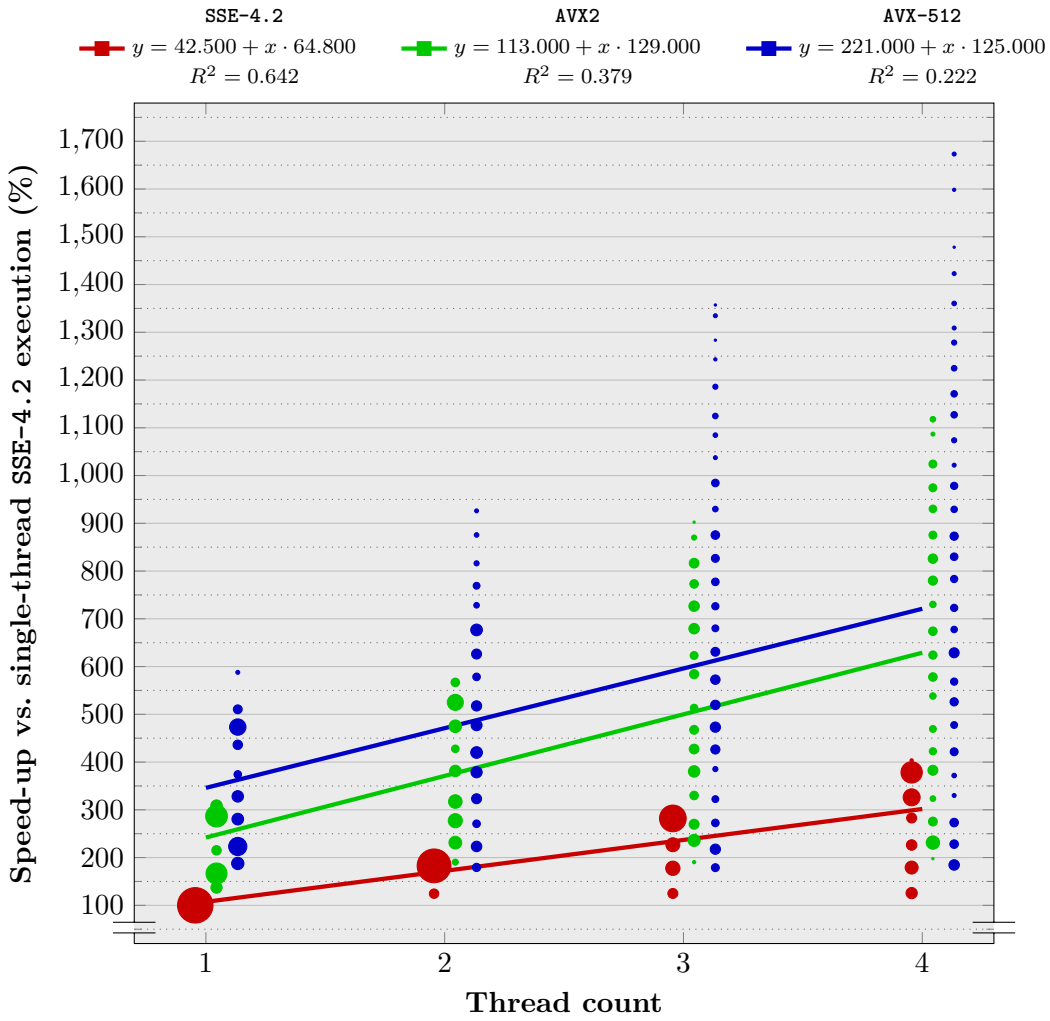


Figure 5.21: Average speed-up of selected platform compared to single-thread `SSE-4.2` execution for all applications except `csv2json` and `u32u8` using `CURR`

This section focuses on answering `Q2`. Because of how `Pablo` leverages SIMD instructions, Parabix programs naturally benefit from improved hardware — more so than traditional "byte-at-a-time" text-parsing applications. Since the rest of this chapter consists of comparison studies between program variants, this aspect is indiscernible without carefully studying Appx. D so we summarize the impact hardware has on `CURR` in Figure 5.21. We exclude `csv2json` and `u32u8` because a dominant component — the `pext` instruction — must be simulated with Warren's parallel-prefix compression algorithm [112] on the `SSE-4.2`, resulting in a somewhat misleading 6 – 23× speed up when comparing `AVX-512` to `SSE-4.2`. For each configuration, every data point in Figure 5.21 is relative to the single thread cost on `SSE-4.2`; i.e., $\frac{\texttt{CYC}(z, TC=x)}{\texttt{CYC}(\texttt{SSE-4.2}, \texttt{TC=1})}$. Like with the speedup plot in §5.2, to highlight the density of the data points, we bin the values in 50% intervals. The percentage of the observed values

that fall into each bin dictates the size its position on the plot is determined by geometric mean of its binned values and the regression lines are computed from the raw data. Based on the regression formulas, we observe an average four-thread performance boost of **2.1**× and a **2.4**× for `AVX2` and `AVX-512`, respectively, compared to `SSE-4.2`.

5.4 Comparison against 2017 version

Here we compare the current version of Parabix against the version dated 8/8/2017, corresponding with Dan Lin's last repository submission. Her version primarily targeted LLVM 3.8 whereas the current framework targets LLVM 6.0 and later versions. To make this study fairer, `2017` was upgraded to support the LLVM 12.0 API. This modified version is located at cs-git-research.cs.surrey.sfu.ca/cameron/parabix-devel/-/tree/2017-comparison. Except for incorporating PAPI instrumentation, no other changes were to `2017`'s pipeline generation or JIT-execution phases. Even so, this evaluation comes with several caveats:

1. `2017` did not support `PopCount` rates nor `LookAhead` (or other) attributes and lieu of `Bounded` rates, it provided `MaxRatio` rates that were relative to some other port, leading to more complicated design in some cases.

2. `2017` does not have a concrete notion of segment length. Although it is precisely controlled in `CURR`, we approximate it in `2017` by increasing the size of the streamset buffers and the output rate of the source kernel.

3. Since 2017, there have been significant changes to all of the RE, `Pablo` and LLVM-IR compilation functions in the framework. `icgrep` is the most affected by these changes since the `2017` system predates much of the focus on UNICODE support.

Because of caveat 3, we reject `icgrep` as a comparison case and focus this section on the remaining applications supported by both `2017` and `CURR`: `base64`, `editd` and `u8u16`. In Figure 5.22, we present the summary of our findings for this section. Like with the speed up plots of §5.2, Figure 5.22 is a weighted scatter plot where we aggregate the reduction-values for `CURR` and `2017`, selecting the ranked best-ranked values for `CURR` and minimal values for `2017`, distinguishing them only by platform and thread count. The observed reduction-values are binned into 5% intervals and the size of each plotted value corresponds with the square root of the number of observations in each bin. We consider each application individually in the following subsections. The main observations from Figure 5.22 are that there is `CURR` is better able to utilize multiple threads and has a clear advantage when using newer architectures. Although `SSE-4.2` appears to benefit more from the `CURR` than `AVX2`, when considering the raw difference, `AVX2` still has a higher throughput.

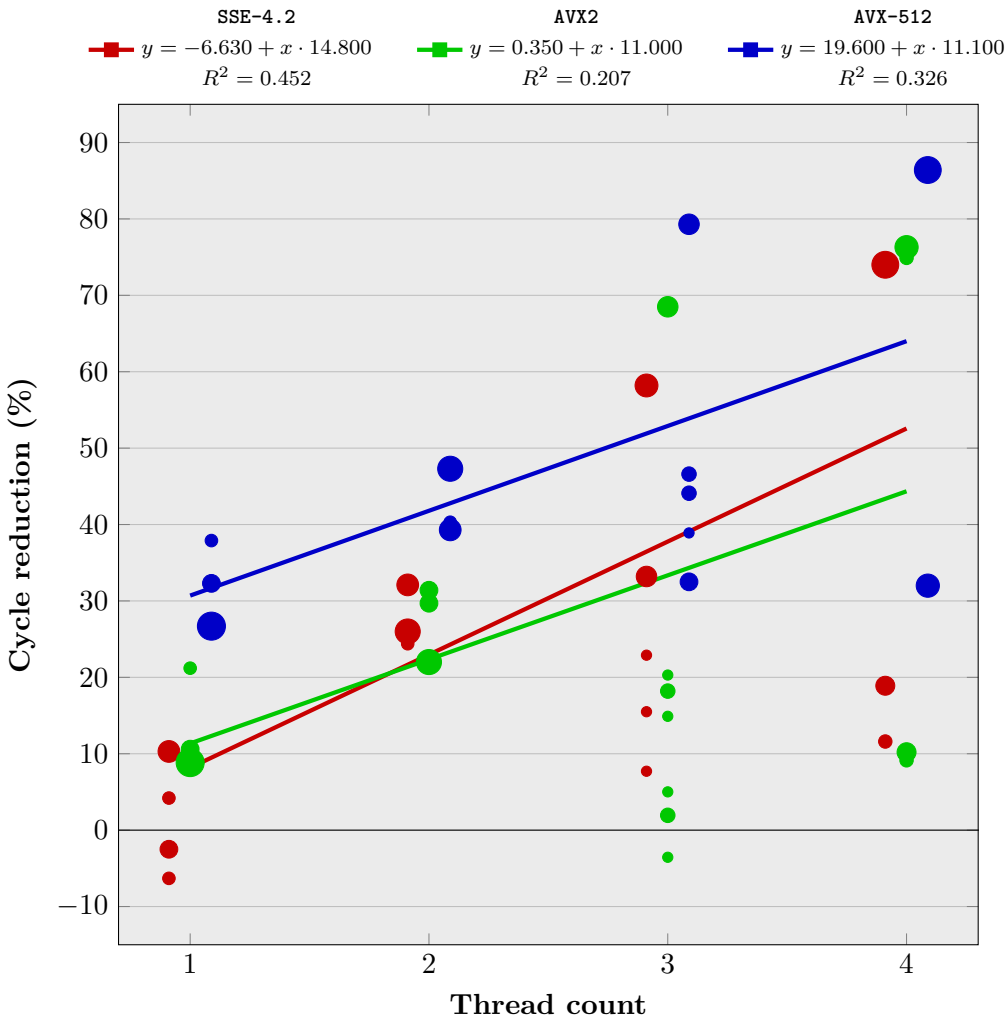


Figure 5.22: Overall reduction in CPU cycles between `2017` and `CURR`

5.4.1 Case study: `base64`

Excluding the hidden boilerplate LLVM-IR necessary for I/O handling, which was moved from the generated kernels in `2017` to the pipeline of `CURR`, the IR generated by the kernels for `base64` within both versions is identical. Note, however, we modified the `base64` program on `2017`. Originally, internal buffers were constructed as `Dynamic` buffers but such buffers are unnecessary for this program. Substituting them with `Static` buffers resulted in a program that whose single threaded was $\sim 2.9\times$ faster than its 4-threaded version. While it is likely that this difference in performance is a result of a design flaw or defect in how `2017` handled `Dynamic` buffers, we swapped them to provide a more useful comparison study. Despite this change, Tbl. 5.8 shows that with an equivalent number of threads, `CURR` is faster than `2017` in all cases and is better able to leverage the newer architectures. Tbl. 5.8 is slightly misleading, however. Shown better in Appx. D.3.1, both frameworks were capable of accelerating the program with two threads but `2017` experiences a drastic slowdown when

utilizing 3 or 4. This is expected because `2017` used strict pipeline parallelism and the cost of the most expensive kernel is $> 40\%$. However, because the majority of the work is statefree, `CURR` continues to accelerate the program past its expected limit.

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	1	32	0.18	0.38	1.80	32	0.07	-0.59	-2.31	32	0.19	0.01	0.00
Arwiki	2	32	0.28	1.78	0.95	32	0.17	8.95	-1.11	32	0.20	3.10	0.02
Arwiki	3	32	0.76	1.72	0.14	32	0.56	8.44	-0.43	32	0.85	3.31	0.34
Arwiki	4	32	1.16	1.85	0.42	32	0.69	6.72	-0.61	32	1.10	2.58	0.01
Enwiki	1	32	0.17	0.39	1.79	32	0.08	-0.66	-2.43	32	0.19	-0.00	0.00
Enwiki	2	32	0.27	1.79	1.03	32	0.18	9.09	-1.17	32	0.21	3.10	0.05
Enwiki	3	32	0.76	1.69	0.08	32	0.56	8.50	-0.42	32	0.83	3.23	0.34
Enwiki	4	32	1.17	1.98	0.38	32	0.68	6.66	-0.62	32	1.15	2.60	0.01
Ruwiki	1	32	0.18	0.39	1.92	32	0.07	-0.35	-2.28	32	0.19	0.01	0.00
Ruwiki	2	32	0.28	1.79	0.93	32	0.17	8.61	-1.09	32	0.20	3.10	0.04
Ruwiki	3	32	0.74	1.71	0.13	32	0.56	8.38	-0.43	32	0.83	3.22	0.34
Ruwiki	4	32	1.14	1.96	0.38	32	0.69	6.70	-0.61	32	1.11	2.58	0.01
Zhwiki	1	32	0.18	0.40	1.89	32	0.07	-0.57	-2.36	32	0.19	0.01	0.00
Zhwiki	2	32	0.28	1.80	1.05	32	0.16	8.66	-1.11	32	0.20	3.10	0.02
Zhwiki	3	32	0.76	1.78	0.18	32	0.56	8.59	-0.42	32	0.88	3.33	0.34
Zhwiki	4	32	1.14	1.92	0.51	32	0.68	6.61	-0.62	32	1.16	2.59	-0.00
		μ	**0.59**	1.46	0.85		**0.37**	5.86	-1.13		**0.59**	2.24	0.09
		σ	0.39	0.62	0.66		0.26	3.79	0.75		0.41	1.32	0.14

Table 5.8: `base64` PAPI normalized difference comparison between `2017` and `CURR`

5.4.2 Case study: `editd`

This case study analyzes the multi edit-distance program wherein multiple strings are matched simultaneously. `editd` was continuously updated with API changes but the underpinning logic has remained untouched since the `2017` version and the design of `editd` prevents it from taking advantage of any of the newer `Pablo` optimization passes. Since the focus of this case study is purely on the performance of the pipeline, we ignore filtering, indexing and prefix-grouping optimizations and all sequential post-processing work. The the compiled pipeline contains 100 pattern matching kernels plus 4 I/O support necessary to read and convert the input data and serialize the result.

In Tbl. 5.9 we vary the edit distance between $1, 2, 4$ and 8 for both programs. We attempted larger edit distances but the `std::vector` holding the position trace for `C++` callback function in the `editdScanMatch` kernel expanded beyond the system memory capacity. Because edit-distance is a fixed user parameter, we incorporate it into the ranking algorithm for `CURR`. Additionally, we omit the 2-thread results since `2017` seg-faulted before completion with that configuration in all test cases.

On `SSE-4.2`, single-threaded execution was slower with `CURR` than `2017` in a few cases. Presumably this is a direct result of the more complex pipeline code since only superficial

differences exist between the `Pablo` code for the `editd` pattern kernels between both versions. In the compiled LLVM-IR, two major differences existed between these versions: 1) in `CURR` the pattern kernels were annotated with `Family` attributes and thus invoked through a function pointer by the pipeline and 2) `Pablo` carry-packing is enabled by default. However, even after disabling both, there was little difference to the resulting measurements.

CONFIG		SSE-4.2				AVX2				AVX-512			
DIST	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
1	1	32	-0.47	-37.90	0.12	32	4.17	-8.70	1.06	32	12.10	-17.50	0.07
1	3	32	6.52	-10.70	-0.09	32	3.46	-0.74	0.84	32	6.17	6.37	1.01
1	4	32	3.93	-7.05	-0.28	32	1.09	-0.49	0.91	32	3.16	4.69	0.70
2	1	32	-4.80	-77.00	-0.06	32	7.86	-35.20	1.19	32	17.90	-29.50	0.19
2	3	32	7.09	-22.80	-0.57	32	4.71	-9.42	4.41	32	8.48	6.29	0.53
2	4	32	6.55	-13.80	-3.31	32	1.93	7.01	-1.78	32	4.77	-2.56	0.93
4	1	32	-15.10	-70.50	-0.23	32	18.00	-75.80	1.77	16	36.00	23.10	0.40
4	3	16	7.23	-23.50	-0.84	16	8.73	-11.00	1.76	16	14.50	8.58	15.40
4	4	16	8.56	-14.20	-2.75	16	4.38	-9.93	5.67	16	9.45	6.39	33.70
8	1	16	29.60	-84.20	-0.93	32	109.00	45.00	-100.00	16	143.00	-34.40	96.60
8	3	16	349.00	-16.00	-112.00	16	40.30	-17.70	89.80	16	50.80	8.95	-1.54
		μ	**36.20**	-34.30	-11.00		**18.50**	-10.60	0.51		**27.80**	-1.78	13.40
		σ	99.40	27.50	32.10		30.60	27.80	40.60		38.90	17.00	28.10

Table 5.9: `editd` PAPI normalized difference comparison between `2017` and `CURR`

Although the performance regression may be related to the increase in L2 and L3 misses on `SSE-4.2`, both `AVX2` and `AVX-512` see a increase in both yet `CURR` executes faster than `2017` in both cases. Notably, `CURR` is 34.2% faster on average on `AVX-512`, reaching speeds up to 37.9% faster than `2017` with edit distance 8. Although `2017` predates the inclusion of the specialized `AVX-512 IRBuilder`, it does support `AVX-2`. Thus 13.2% average performance improvement observed in `AVX2` cannot be attributed to the better compilation support. Further investigation into the cause of these cache misses may lead to future performance fixes on older architectures. One final note, however, is that on `CURR` (and presumably `2017`) between 10% and 25% of the total L3 misses were attributed to the `streamsMerge` kernel, which combines the match results of all 100 patterns. This could be eliminated entirely with in/out streamset memory but that is not currently supported by `CURR`.

5.4.3 Case study: `u8u16`

Conceptually, `u8u16` has remained largely unmodified since 2017 but a substantial optimization to the underlying compression techniques has been made since the `2017` version. To make this comparison fair, the deletion and compression kernels from `2017` were ported to the match the current API and used in lieu of the current version. It should be noted,

however, that this compression approach is known to be incorrect in some cases but equivalently so between the variants except for a few incomplete final-stride cases for the `2017` variant that does not notably affect the total number of output bytes.

In Tbl. 5.10, we study the performance of both versions against the files listed in Tbl. 5.1. Unfortunately, of the tested configurations, `2017` only successfully ran to completion with the two and three-thread versions. Given those data points, we still see that `CURR` executes faster than `2017` in almost every case but exhibits a notable drop in performance on `AVX2`. As Tbl. 5.11 shows, this is because the cost of the most expensive kernel in this version of `u8u16` is considerably higher with `AVX2` than the other instruction sets. Since the `2017` compression algorithm was replaced, no further investigation was made. Despite this, when comparing the best results for `2017` against the ranked result of `CURR` on `AVX-512`, both two and three-thread versions show a consistent 40% to 50% reduction in cycles in almost every case (See Appx. D.3.3 for details.) Interestingly, there is an 59% increase in L3 misses on `SSE-4.2` but a trivial difference on the other platforms. Given that this only represents an increase of 2.3×10^{-4} L3 misses per byte and both `AVX2` and `AVX-512` have 10× the total L3 misses than `SSE-4.2` this is not likely to be a major issue. However, when considered in the context of the `editd` study, this does point to the potential of further improvements on older platforms.

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	2	32	0.87	0.21	-0.22	16	0.36	10.50	-0.07	32	0.71	4.35	0.18
Arwiki	3	32	0.60	0.67	-0.20	32	0.01	5.50	-0.08	32	0.48	3.08	0.26
Enwiki	2	32	0.88	0.10	-0.32	16	0.35	10.40	-0.03	32	0.71	5.21	0.24
Enwiki	3	32	0.60	0.52	-0.29	32	-0.04	4.47	-0.02	32	0.33	3.02	0.16
Ruwiki	2	32	0.88	0.14	-0.29	16	0.36	10.30	-0.01	32	0.73	4.49	0.34
Ruwiki	3	32	0.57	0.63	-0.15	32	0.03	5.94	-0.07	32	0.46	2.79	0.22
Zhwiki	2	32	0.69	0.13	-0.18	16	0.36	10.30	-0.04	32	0.71	4.50	0.25
Zhwiki	3	32	0.62	0.70	-0.17	32	0.05	6.02	-0.06	32	0.45	2.93	0.21
		μ	**0.71**	0.39	-0.23		**0.18**	7.94	-0.05		**0.57**	3.80	0.23
		σ	0.13	0.25	0.06		0.17	2.50	0.02		0.15	0.88	0.05

Table 5.10: `u8u16` PAPI normalized difference comparison between `2017` and `CURR′`

TC	SSE-4.2	AVX2	AVX-512
1	43.3	49.8	43.3
2	45.8	51.3	45.8
3	55.7	66.9	55.7
4	67.1	75.4	67.1

Table 5.11: Average relative cost (%) of most expensive `u8u16` kernel for `CURR′`

5.5 Comparison against 2020 version

In the case studies in §5.4, we compared `CURR` against `2017` but because of the `2017`'s limitations, it is incapable of compiling the more complex programs without significant modification, specifically those that may benefit from partitioning. To assess the more complex programs, we extracted the pipeline compiler from the September 24, 2020 revision, which we herein label the `2020` version, only slightly modifying it to match the internal API of `CURR`. This version was inserted into the current framework as a secondary pipeline compiler, limiting the differences in the JIT-ed code to only those within the `PipelineKernel`. `2020` is located at cs-git-research.cs.surrey.sfu.ca/cameron/parabix-devel/-/tree/2020-comparison

We selected `2020` since since later revisions incorporated some notion of partitions whereas `2020` is closer to the `2017` pipeline except that it took advantage of caching, contained significantly more functionality, and, like `CURR`, was responsible for almost-all of the memory management at JIT-execution time. These case studies primarily focus on answering secondary question `Q3` as the pipelines for both `CURR` and `2020` were optimized for their given tasks. Although most programs show a clear improvement with the `CURR`, `icgrep -colours=never` often favours `2020`. Since combining all of the results into a single figure loses this nuance, we separate the results into two groups, presented in Figure 5.23 and Figure 5.24, respectively.

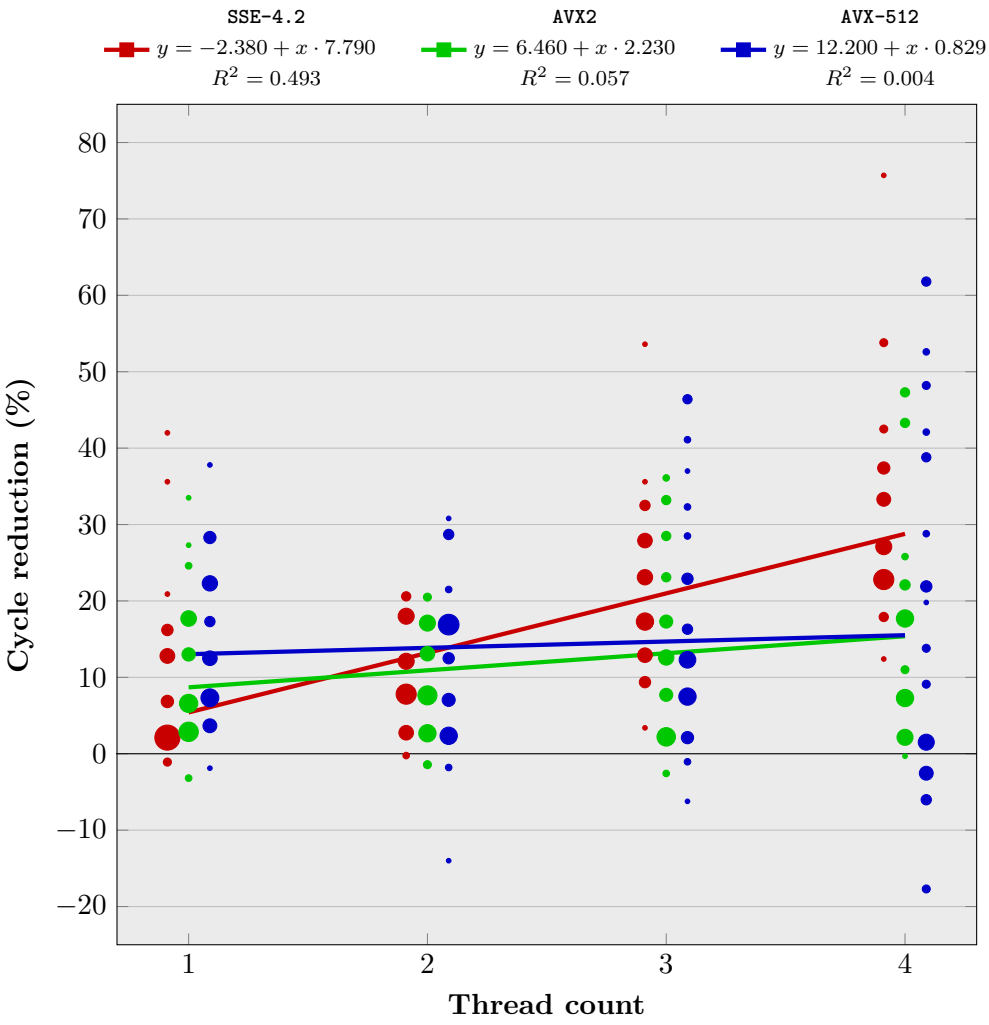


Figure 5.23: Overall reduction in CPU cycles between `2020` and `CURR` with all applications except `icgrep -colours=never`

As Figure 5.23 shows, for the set of "well behaved" programs on average we have a consistent 5.4 - 13.1% reduction in single-threaded cycle cost. However, the reduction spread varies as the thread count increases, especially for `AVX-512`. The absolute difference, however, is less substantial since, as Figure 5.21 indicates, the total cost of each program reduces considerably with newer hardware.

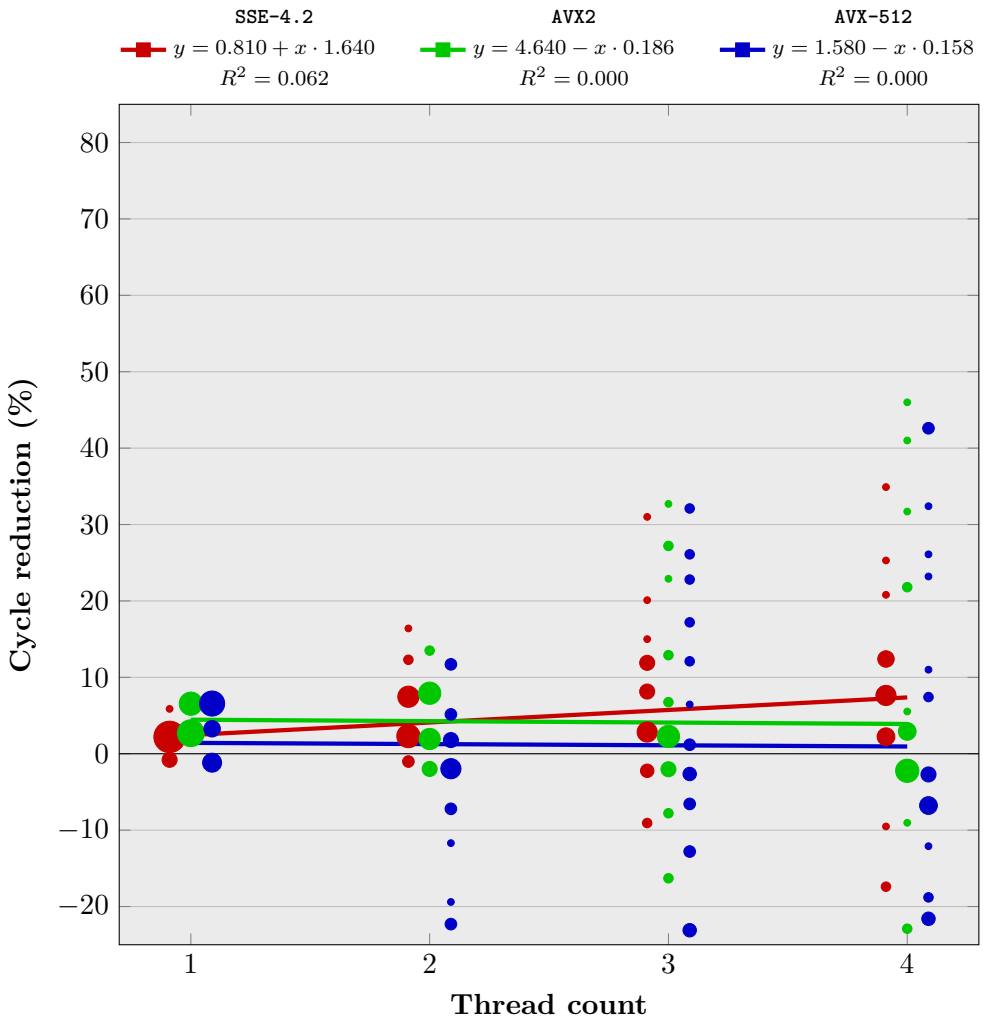


Figure 5.24: Overall reduction in CPU cycles between `2020` and `CURR` for `icgrep -colours=never`

In `icgrep -colours=never`, shown in Figure 5.24, we see that even though the average performance still favours `CURR`, a significantly higher proportion of the results shows a performance regression. We discuss this case in more detail in the next subsection.

5.5.1 Case study: `icgrep -colours=never`

In this case study, we consider the seven REs, listed in Tbl. 5.1 with the `-colours=never` setting but focus on the `@` and `Email` expressions as they are the only ones that have consistent performance regressions on all three platforms. A common trait of all of these is that there is slightly better single thread performance on `CURR` but worse multi-thread performance in both `AVX2` and `AVX-512`. As shown in §5.2.5, `icgrep -colours=never` scales poorly using multiple threads with `CURR`. The main reason is that the average relative cost of the most expensive kernel is higher with `CURR` than `2020`. For example, the cost in a single-threaded execution of `@` and `Email` with Arwiki on `AVX2` is 47.8% and 51.2% for `CURR` but

only 42.0% and 45.4% for `2020`. This explains the slowdown between these variants, shown in Figure 5.12 and 5.13, but does not indicate why. Although cache misses are obviously an issue for the `Email` case, it is less clear whether this is the culprit for `@` on `SSE-4.2`.

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	1	32	0.02	-0.02	-0.18	32	0.04	0.06	0.07	32	0.03	-0.00	-0.00
Arwiki	2	32	0.03	-0.02	-0.08	32	0.03	-0.12	0.25	32	-0.06	-0.00	0.02
Arwiki	3	32	0.01	-0.12	-0.16	32	0.00	-0.03	0.26	32	-0.08	-0.00	0.02
Arwiki	4	32	0.02	-0.10	-0.24	32	0.00	-0.00	0.24	32	-0.08	0.02	0.03
Enwiki	1	32	0.02	0.01	0.05	32	0.03	0.06	0.04	32	0.03	-0.00	-0.00
Enwiki	2	32	0.01	-0.09	-0.08	32	0.02	-0.07	0.16	32	-0.07	0.00	0.02
Enwiki	3	32	0.01	-0.12	-0.13	32	0.01	-0.05	0.21	32	-0.08	-0.02	0.01
Enwiki	4	32	0.03	-0.10	-0.20	32	-0.01	0.01	0.23	32	-0.08	0.02	0.04
Ruwiki	1	32	0.03	0.02	0.16	32	0.04	0.04	0.04	32	0.03	-0.00	0.00
Ruwiki	2	32	0.05	-0.12	-0.13	32	0.01	-0.08	0.13	32	-0.06	-0.00	0.02
Ruwiki	3	32	0.03	-0.13	-0.14	32	0.00	-0.03	0.21	32	-0.07	-0.01	0.01
Ruwiki	4	32	0.02	-0.10	-0.21	32	-0.01	0.00	0.17	32	-0.07	0.02	0.02
Zhwiki	1	32	0.03	-0.00	0.03	32	0.04	0.05	0.07	32	0.04	-0.00	0.02
Zhwiki	2	32	0.02	-0.10	-0.07	32	0.02	-0.08	0.21	32	-0.07	-0.01	0.02
Zhwiki	3	32	0.03	-0.13	-0.16	32	0.01	-0.01	0.34	32	-0.08	-0.00	0.02
Zhwiki	4	32	0.02	-0.10	-0.23	32	-0.01	0.03	0.22	32	-0.08	0.03	0.06
		μ	**0.02**	-0.08	-0.11		**0.01**	-0.01	0.18		**-0.05**	0.00	0.02
		σ	0.01	0.05	0.11		0.02	0.05	0.08		0.05	0.01	0.02

Table 5.12: `icgrep` PAPI normalized difference comparison between `2020` and `CURR` with arguments Expression=@, Colours=never

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	1	32	-0.01	-0.47	-1.91	32	0.02	0.41	5.73	32	-0.03	-0.00	-0.00
Arwiki	2	32	0.09	-0.51	-0.86	32	0.02	-0.41	3.23	32	0.02	-0.12	0.02
Arwiki	3	32	-0.05	-1.48	-0.54	32	-0.12	-2.47	2.20	32	0.07	-0.25	0.02
Arwiki	4	32	-0.15	-1.71	-0.46	32	-0.18	-2.44	1.65	32	-0.03	-0.28	0.02
Enwiki	1	32	0.04	0.07	0.27	32	0.04	0.16	1.08	32	-0.00	-0.00	0.00
Enwiki	2	32	0.07	-0.08	0.07	32	0.00	-0.18	0.72	32	0.02	-0.05	0.04
Enwiki	3	32	0.09	-0.14	-0.01	32	0.12	-0.46	0.75	32	0.14	-0.07	0.02
Enwiki	4	32	0.12	-0.17	-0.05	32	0.11	-0.72	0.51	32	0.12	-0.08	0.04
Ruwiki	1	32	-0.01	-0.48	-1.91	32	0.02	0.44	5.99	32	-0.03	-0.00	0.00
Ruwiki	2	32	0.19	-0.48	-0.90	32	0.06	-0.39	3.29	32	0.08	-0.12	0.02
Ruwiki	3	32	-0.13	-1.84	-0.52	32	-0.17	-3.03	2.25	32	-0.01	-0.38	0.01
Ruwiki	4	32	-0.28	-2.06	-0.45	32	-0.23	-2.95	1.65	32	-0.15	-0.37	0.01
Zhwiki	1	32	-0.07	-0.45	-1.79	32	0.01	0.53	6.34	32	-0.03	-0.00	-0.00
Zhwiki	2	32	0.01	-1.01	-0.85	32	0.11	-0.94	3.52	32	0.12	-0.16	0.03
Zhwiki	3	32	-0.20	-2.52	-0.57	32	-0.09	-3.37	2.29	32	-0.03	-0.46	0.02
Zhwiki	4	32	-0.34	-2.65	-0.46	32	-0.11	-2.99	1.70	32	-0.18	-0.45	0.03
		μ	**-0.04**	-1.00	-0.68		**-0.03**	-1.18	2.68		**0.00**	-0.17	0.02
		σ	0.14	0.88	0.66		0.11	1.38	1.83		0.09	0.16	0.01

Table 5.13: `icgrep` PAPI normalized difference comparison between `2020` and `CURR` with arguments Expression=Email, Colours=never

These programs are simple enough that neither partitioning or thread-local memory should be a factor nor does either variant experience buffer expansions. Additionally, the strides processed per segment for both variants are identical. However, one issue that is hidden here is the use of linear buffers: `2020` only supports circular buffers but `CURR` allows either. In this particular case, two of the four streamset were constructed as linear buffers in `CURR`. Because the data-access pattern dictated by the I/O rates indicates that a copyback is not required by linear buffers (aside from the final segment in the case that the input is an exact multiple of the segment-length) they were left as linear to avoid the modulus calculations. A cursory inspection indicated this is true for the single-threaded case but not always true in the multi-threaded ones. The issue is that in a multi-threaded scenario, a streamset producer does not always have the most up to date information about the data consumption state from other threads. The amount of data being copied on average does not indicate this should be a major issue but the results clearly hint that it is. Further investigation into this case may reveal improvements into memory layout and data usage strategies.

5.6 Comparison against hybrid pipeline

The hybrid-pipeline model permits a single fixed-code thread within a fixed-data pipeline. We discuss our implementation of this model in §4.5. Based on §5.2, we selected `icgrep`

`-colours=never` since the dominate kernel, `ScanMatchKernel`[4], accounted for between 30% and 86% of the total work depending on the configuration.

The goal of this study is to determine whether isolating the dominant kernel permits us to better leverage thread parallelism. Although both program variants use the `CURR` framework, we label the programs with hybrid threading enabled as `HYBRID`. Because each of these programs generally do not see an improvement in `CURR` beyond two or three threads and often see worse performance with more, we also modify the formulation of Figure 5.25 compared to the prior weighted scatter plots. When contrasting `CURR` and `HYBRID`, for each identical configuration we compute $^{\texttt{CURR.CYC}-\texttt{HYBRID.CYC}}/_{\texttt{CURR.CYC}}$ but select the lowest value of `CURR.CYC` that uses no more threads than `HYBRID.CYC`, instead of an equivalent number. We also ignore ranking and instead select the best segment-length setting for both variants. Our reason for these changes is that we want to avoid showing reductions for the `HYBRID` that do not are not improvements for overall performance.

Unfortunately, `HYBRID` exhibits worse overall performance. The few cases that do exhibit actual improvements are limited to the cases using the REs `URIOrEmail` and `\X`, both of which generate a greater number of output bytes compared to other REs. While `\X` could be excused by being a pathological case, it is interesting that `URIOrEmail` also sees an improvement. As shown in Appx. D.1.5, the cost of `ScanMatchKernel` for `URIOrEmail` is often less than other REs. Based on this experiment, we conclude that naively selecting the most expensive kernel is not a sufficient strategy. It should be noted, however, is that Mastoras relied on work stealing to ensure that hybrid threads always remained busy [78]. Supporting this requires some form of dynamic scheduling but if this is incorporated into the framework, it may be beneficial to reassess this concept in the future.

[4]The `ScanMatchKernel` gathers the line break aligned positions for the matching lines and reports them to the end user.

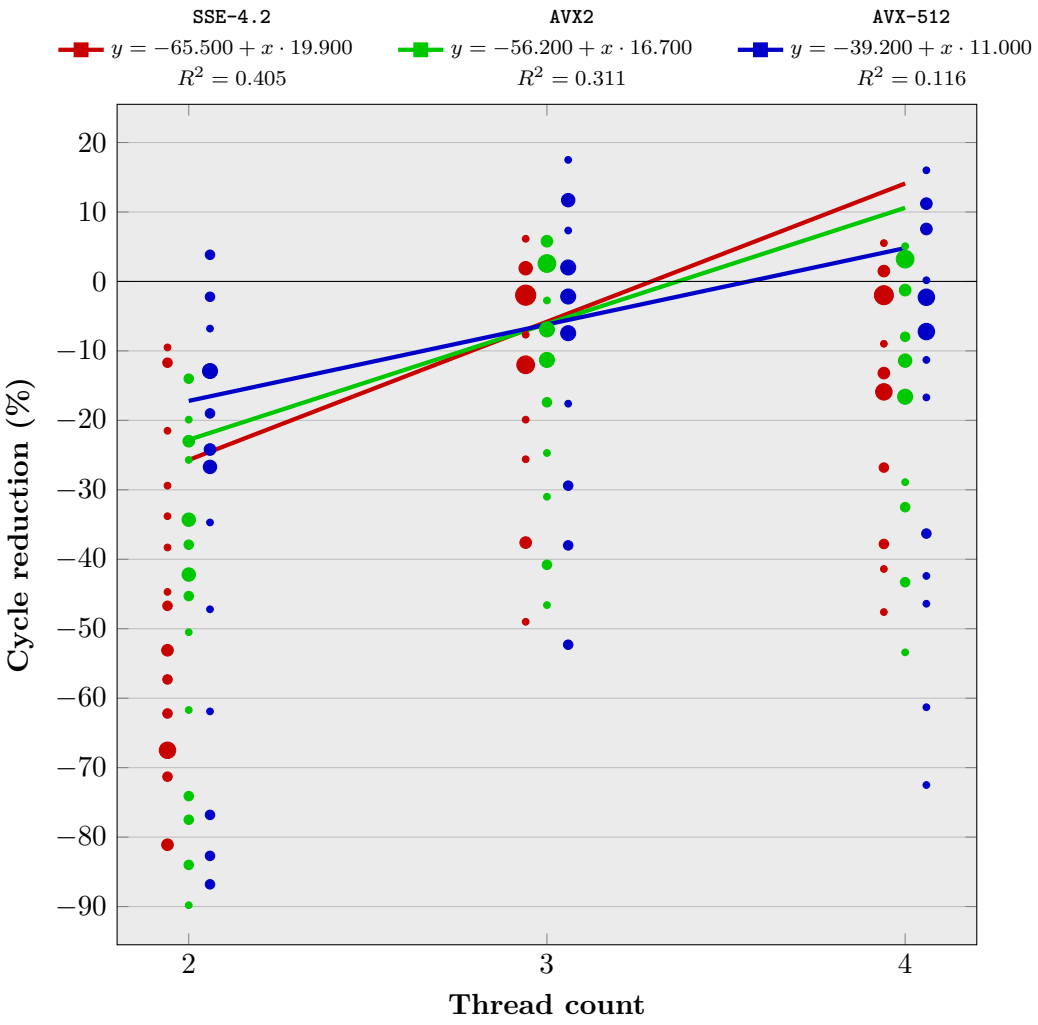


Figure 5.25: Overall reduction in CPU cycles between `CURR` and `HYBRID` by program

5.7 Comparison against non-Parabix applications

In this section we evaluate `CURR` against a variety of state of the art non-Parabix programs. Because these programs are limited to single-threaded execution and do not have a notion of segment-length, we limit `CURR` to one-thread with the best ranked segment-size for each program. The purpose of this study is to determine whether benchmark applications used in this study are comparable to state of the art programs even before applying multi-thread acceleration. It should be noted that the vast majority of the cases here have substantively fewer L2 and L3 misses than `CURR`. This is primarily due to the fact well designed sequential parsers tend to have predictable cache usage patterns but in trade for that predictability, they have a higher potential for branch misses.

5.7.1 Case study: `base64` vs. `fastbase64`

The `fastbase64` program is a general implementation of Muła and Lemire's work [84] on fast base64 encoding/decoding; it can be located at github.com/aklomp/base64. Although this version only supports up to `AVX-2` instructions, `fastbase64` is considerably faster than `CURR` in every case. One obvious factor is the three to four orders of magnitude difference in L3 misses. Despite the fact ∼98% of the executed instructions in `base64` are classified as useful work (as opposed to pipeline logic or buffer management costs), the sheer number of misses more than explains the difference in performance results.

When comparing the two programs, there are a few major differences: `base64` is a five kernel program in which each kernel processes data derived from 4/16/32 KB chunk of file data per segment but the work done by every kernel is relatively small. `fastbase64`, however, has a very tight input to output encoding loop, converting file data in ∼1MB chunks per segment. When comparing the two programs, the longer instruction chain in `fastbase64` greatly helps to improve performance by better absorbing the cost of memory latency. Obviously, we could merge several kernels into a single one but doing so would limit the ability to multithread the program. Directly replicating the tight input to output loop would be equivalent to executing a factored SDF schedule. In this particular case, this may be a reasonable approach to consider (given that most of `base64`'s kernels are statefree) but generally speaking, a factored program would either be limited to only parallelization of the last subsegment iteration or — as §3.1.2 discusses — see a massive increase the kernel synchronization and state memory transfer cost. More analysis on this case may help further refine the segment-based processing model for simple programs. Note: future studies should consider adding the `base64-avx512` benchmark from github.com/WojciechMula/base64-avx512, which is based on the `AVX-512` improvements to `fastbase64` discussed in [85].

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	1	32	-0.98	-0.41	-3.01	32	-0.48	3.99	-19.40	32	-0.21	2.09	-15.60
Enwiki	1	32	-0.98	-0.41	-3.02	32	-0.48	4.09	-19.20	32	-0.22	2.09	-15.60
Ruwiki	1	32	-0.98	-0.40	-2.97	32	-0.48	4.19	-19.30	32	-0.21	2.09	-15.60
Zhwiki	1	32	-0.98	-0.38	-2.81	32	-0.48	4.00	-19.40	32	-0.21	2.09	-15.60
		μ	**-0.98**	-0.40	-2.95		**-0.48**	4.07	-19.30		**-0.21**	2.09	-15.60
		σ	0.00	0.01	0.08		0.00	0.08	0.09		0.00	0.00	0.00

Table 5.14: `base64` PAPI normalized difference comparison between `FB64` and `CURR`

5.7.2 Case study: `csv2json` vs. sequential `csv2json`

Sequential `csv2json` is a `C`-based CSV to JSON converter, designed for `UTF-8` data; it is located at github.com/webcarrot/csv2json. A review of the publicly-available CSV to JSON converters on `github` found this was the most efficient one for our test cases. Parabix

`csv2json` relies on the `pext` instruction but this must be simulated using Warren's parallel-prefix compression algorithm [112] on the `SSE-4.2`. Unsurprisingly, on `SSE-4.2`, `csv2json` performs worse than its sequential counterpart but on both `AVX2` and `AVX-512`, Parabix `csv2json` is significantly faster in each case.

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Census	1	32	0.86	-2.54	-0.52	32	17.10	-40.50	1.12	16	19.90	-8.39	-0.01
Fin	1	32	-20.70	-3.57	-1.55	32	4.86	-49.20	-14.30	16	6.72	-13.00	-5.37
Trade	1	32	-16.40	-3.01	-1.06	32	4.86	-46.80	-0.25	16	6.40	-11.60	-0.27
		μ	**-12.10**	-3.04	-1.04		**8.94**	-45.50	-4.48		**11.00**	-11.00	-1.88
		σ	9.33	0.42	0.42		5.76	3.63	6.97		6.29	1.93	2.47

Table 5.15: `csv2json` PAPI normalized difference comparison between `SEQ` and `CURR`

5.7.3 Case study: `icgrep` vs. `ugrep`

`ugrep` is a standalone replacement for GNU/BSD grep that supports `UNICODE` text searches and is reported to be faster than `ripgrep` for single file queries; it can be located at github.com/Genivia/ugrep. To get consistent measurements from both `ugrep` and `icgrep`, `ugrep` was modified to prevent it from silently enabling quiet mode when outputting data to `/dev/null` and each query was executed using the `-a` flag, ensuring it treated all input as text. The `-a` flag is classified as an optimization for `ugrep` as it bypasses the `UTF` verification phase of the program. In the following tables, we separate the results by colourization mode. Because of the design of their search engines, there is a trivial difference in performance `ugrep` between the `always` and `never` modes but a substantial difference in `icgrep`.

Without colourized output, `icgrep` outperforms `ugrep` on all three tested architectures except for the '`@`' expression on `SSE-4.2`. The relative degree to which `icgrep` exceeds `ugrep` increases as the quality of the hardware increases. With colourization, however, `ugrep` tends to outperform `icgrep` with simple queries but can take several orders of magnitude more time for the complex ones.

Even with colourization, on the `AVX-512` `icgrep` is significantly faster than `ugrep` except for the '`@`' query. Although `CURR`'s initialization cost might be higher than `ugrep`'s for this RE, it is a fixed cost in both cases and thus cannot explain the difference. This case ought to be investigated further to determine what improvements to `icgrep` and the `PipelineCompiler` are possible.

Arguments: `-colours=always`

CONFIG			SSE-4.2				AVX2				AVX-512			
EXPRESSION	FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
@	Arwiki	1	32	-4.20	-0.21	-3.86	32	-0.99	1.14	-16.40	32	-0.45	-1.36	-15.80
@	Enwiki	1	32	-4.19	-0.16	-3.80	32	-0.98	1.35	-16.30	32	-0.45	-1.52	-15.70
@	Ruwiki	1	32	-4.21	-0.20	-3.76	32	-1.01	1.33	-16.50	32	-0.44	-1.33	-15.80
@	Zhwiki	1	32	-4.26	-0.22	-3.70	32	-1.02	0.49	-16.60	32	-0.47	-1.45	-16.00
Date	Arwiki	1	32	-3.07	-1.27	-4.22	32	1.82	-3.29	-23.40	32	1.69	-1.84	-17.20
Date	Enwiki	1	32	1.67	-0.68	-3.76	32	4.98	2.01	-21.50	32	5.90	-1.34	-15.70
Date	Ruwiki	1	32	-3.35	-1.35	-4.23	32	1.68	-4.10	-23.30	32	1.53	-1.90	-17.30
Date	Zhwiki	1	32	-3.40	-1.54	-4.28	32	1.80	-5.56	-23.90	32	1.71	-2.08	-17.70
Email	Arwiki	1	32	443.00	-0.96	-1.74	32	372.00	-3.07	-22.30	32	393.00	-1.03	-15.70
Email	Enwiki	1	32	546.00	-0.73	-1.63	32	442.00	-1.72	-22.20	32	467.00	-1.16	-15.60
Email	Ruwiki	1	32	361.00	-0.83	-1.79	32	302.00	-1.97	-22.20	32	318.00	-1.18	-15.80
Email	Zhwiki	1	32	1,600.00	-0.87	-1.79	32	1,260.00	-2.04	-22.20	32	1,340.00	-1.10	-15.80
URIOrEmail	Arwiki	1	32	254.00	-4.53	-7.46	32	219.00	-29.80	-39.10	32	233.00	-5.99	-29.30
URIOrEmail	Enwiki	1	32	544.00	-0.83	-1.75	32	441.00	-3.06	-22.10	32	462.00	-1.10	-15.60
URIOrEmail	Ruwiki	1	32	259.00	-4.31	-6.86	32	220.00	-28.10	-37.80	32	237.00	-5.69	-28.30
URIOrEmail	Zhwiki	1	32	1,270.00	-4.45	-7.74	32	999.00	-30.10	-41.00	32	1,060.00	-6.40	-30.70
Xquote	Arwiki	1	32	-14.40	-4.99	-7.48	32	-1.67	-34.70	-39.70	32	1.30	-6.49	-29.90
Xquote	Enwiki	1	32	-9.61	-1.51	-1.84	32	-1.95	-11.20	-22.40	32	0.31	-1.54	-16.10
Xquote	Ruwiki	1	32	-13.90	-4.47	-6.51	32	-1.84	-30.90	-36.50	32	1.07	-5.47	-27.40
Xquote	Zhwiki	1	32	-14.30	-4.62	-7.36	32	-1.60	-33.20	-40.00	32	1.34	-6.22	-30.00
\X	Arwiki	1	32	319.00	-11.00	-21.90	32	311.00	-98.10	-127.00	32	317.00	-32.60	-92.30
\X	Enwiki	1	32	76.50	-12.10	-26.70	32	118.00	-103.00	-149.00	32	115.00	-35.10	-107.00
\X	Ruwiki	1	32	452.00	-10.20	-18.80	32	422.00	-90.30	-102.00	32	430.00	-27.80	-75.90
\X	Zhwiki	1	32	288.00	-11.50	-21.60	32	302.00	-104.00	-120.00	32	311.00	-32.80	-88.30
\p{Greek}	Arwiki	1	32	-5.81	-0.94	-1.92	32	-0.15	-2.18	-22.40	32	0.87	-1.39	-15.80
\p{Greek}	Enwiki	1	32	-5.53	-0.70	-1.76	32	-0.09	-1.33	-22.40	32	0.88	-1.39	-15.80
\p{Greek}	Ruwiki	1	32	-5.95	-1.05	-2.08	32	-0.18	-2.65	-22.80	32	0.89	-1.54	-16.10
\p{Greek}	Zhwiki	1	32	-6.34	-1.02	-2.02	32	-0.39	-2.52	-22.80	32	0.74	-1.56	-16.20
			μ	**226.00**	-3.11	-6.51		**193.00**	-22.20	-39.80		**203.00**	-6.79	-29.60
			σ	388.00	3.65	6.82		307.00	33.60	36.00		324.00	10.50	25.90

Table 5.16: `icgrep -colours=always` PAPI normalized difference comparison between `UGREP` and `CURR`

Arguments: `-colours=never`

CONFIG			SSE-4.2				AVX2				AVX-512			
EXPRESSION	FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
@	Arwiki	1	32	-0.02	-0.23	-4.36	32	0.21	3.09	-16.00	32	0.21	-1.34	-15.60
@	Enwiki	1	32	-0.01	-0.20	-4.11	32	0.21	2.92	-16.10	32	0.21	-1.39	-15.60
@	Ruwiki	1	32	-0.01	-0.23	-4.24	32	0.21	3.12	-16.00	32	0.22	-1.32	-15.60
@	Zhwiki	1	32	0.01	-0.19	-3.97	32	0.22	2.55	-16.10	32	0.22	-1.34	-15.70
Date	Arwiki	1	32	2.11	-0.22	-2.19	32	3.35	1.70	-22.70	32	2.57	-1.34	-16.60
Date	Enwiki	1	32	6.00	-0.12	-1.53	32	6.28	3.57	-21.50	32	6.64	-1.38	-15.60
Date	Ruwiki	1	32	1.98	-0.21	-2.11	32	3.25	1.18	-22.60	32	2.43	-1.26	-16.50
Date	Zhwiki	1	32	2.24	-0.20	-2.16	32	3.46	0.81	-22.70	32	2.69	-1.41	-16.60
Email	Arwiki	1	32	449.00	-0.32	-2.81	32	373.00	3.73	-17.50	32	393.00	-1.11	-15.60
Email	Enwiki	1	32	553.00	-0.31	-2.61	32	444.00	3.63	-17.50	32	467.00	-1.18	-15.60
Email	Ruwiki	1	32	368.00	-0.31	-2.76	32	303.00	3.68	-17.50	32	320.00	-1.11	-15.60
Email	Zhwiki	1	32	1,600.00	-0.24	-2.68	32	1,270.00	3.58	-17.60	32	1,340.00	-0.98	-15.50
URIOrEmail	Arwiki	1	32	270.00	-0.69	-5.68	32	222.00	-1.05	-33.30	32	235.00	-2.04	-24.70
URIOrEmail	Enwiki	1	32	552.00	-0.04	-1.51	32	444.00	3.37	-22.40	32	463.00	-1.12	-15.60
URIOrEmail	Ruwiki	1	32	277.00	-0.59	-5.18	32	224.00	-0.89	-32.00	32	239.00	-1.91	-23.80
URIOrEmail	Zhwiki	1	32	1,290.00	-0.65	-5.83	32	1,000.00	-0.75	-33.90	32	1,060.00	-2.02	-25.30
Xquote	Arwiki	1	32	2.61	-0.85	-5.58	32	2.81	-1.62	-33.00	32	3.84	-2.24	-24.40
Xquote	Enwiki	1	32	0.11	-0.20	-1.59	32	0.70	0.50	-22.30	32	1.59	-1.39	-15.70
Xquote	Ruwiki	1	32	2.14	-0.73	-5.01	32	2.36	-1.31	-31.40	32	3.42	-2.08	-23.10
Xquote	Zhwiki	1	32	2.47	-0.83	-5.49	32	2.86	-1.59	-32.70	32	3.91	-2.31	-24.30
\X	Arwiki	1	32	338.00	-4.39	-18.10	32	317.00	-18.60	-68.20	32	320.00	-5.27	-54.40
\X	Enwiki	1	32	97.20	-4.22	-15.40	32	126.00	-18.20	-60.40	32	120.00	-5.39	-47.90
\X	Ruwiki	1	32	469.00	-3.82	-16.20	32	425.00	-17.30	-63.30	32	432.00	-4.70	-50.00
\X	Zhwiki	1	32	335.00	-4.61	-17.90	32	315.00	-19.70	-69.50	32	317.00	-5.55	-55.10
\p{Greek}	Arwiki	1	32	1.27	-0.14	-1.45	32	1.78	3.17	-22.40	32	1.88	-1.31	-15.60
\p{Greek}	Enwiki	1	32	1.43	-0.14	-1.43	32	1.82	2.30	-22.30	32	1.87	-1.38	-15.60
\p{Greek}	Ruwiki	1	32	1.28	-0.17	-1.68	32	1.81	2.27	-22.60	32	1.93	-1.31	-15.80
\p{Greek}	Zhwiki	1	32	0.88	-0.15	-1.64	32	1.61	2.26	-22.50	32	1.81	-1.34	-15.70
			μ	**237.00**	-0.89	-5.19		**196.00**	-1.20	-29.10		**205.00**	-2.02	-22.80
			σ	388.00	1.40	5.01		308.00	7.26	15.90		324.00	1.36	12.40

Table 5.17: `icgrep -colours=never` PAPI normalized difference comparison between `UGREP` and `CURR`

5.7.4 Case study: `u8to16` and `u32to8` vs. `utf8lut`

The `utf8lut` library is a `UTF-8` conversion utility related on the work of Lemire and Muła [73]. it can be located at github.com/stgatilov/utf8lut. It is a transcoder that leverages the `SSE` instruction set and is capable of converting `UTF-8` to `UTF-16` or `UTF-32` and v.v.. Interestingly, for `UTF-8` to `UTF-16` transcoding `utf8lut` is slower on `SSE-4.2` and `AVX2` yet faster on `AVX-512`. It should be noted that the version of `u8u16` currently supported by the framework is known to be slower than the original algorithm described in by Cameron in [23], which used a cheaper "zero extension" on vectors of ASCII bytes. Experimental work on conditional kernel execution similar to the if-then-else conditional schema of [42] was ongoing at the writing of this dissertation. With `UTF-32` to `UTF-8`, however, `utf8lut` is significantly faster than `u32u8` on all three platforms. Unlike `u8u16`, `u32u8` was not designed as a high-performance application and instead as a stress-test for `PopCount` rates and

current compression techniques so this result is not completely unexpected. Even so `u32u8` remains an interesting case for further study. We recommend that future studies consider benchmarking against the `simdutf` library by Lemire and Muła, as the authors have stated that one is often superior in performance compared to `utf8lut` and support for AVX-512 is currently in development.

`UTF-8 to UTF-16`

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	1	32	-1.52	-0.22	0.23	16	-0.52	1.80	3.77	4	0.81	1.48	0.11
Enwiki	1	32	-1.05	-0.99	-0.31	16	-0.21	-2.59	3.82	4	1.25	0.23	0.18
Ruwiki	1	32	-1.73	0.51	1.41	16	-0.71	2.43	4.06	4	0.66	1.57	0.15
Zhwiki	1	32	-1.92	-0.13	0.56	16	-0.91	0.04	3.95	4	0.57	0.79	0.22
		μ	**-1.55**	-0.21	0.47		**-0.59**	0.42	3.90		**0.82**	1.02	0.17
		σ	0.32	0.53	0.63		0.26	1.95	0.12		0.26	0.55	0.04

Table 5.18: `u8u16` PAPI normalized difference comparison between `UTF8LUT` and `CURR`

`UTF-32 to UTF-8`

CONFIG		SSE-4.2				AVX2				AVX-512			
FILE	TC	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$	SL	CYC	L2M $\times 10^{-2}$	L3M $\times 10^{-3}$
Arwiki	1	32	-6.29	-2.19	-3.19	16	-1.23	-9.74	5.19	4	-0.56	0.01	0.02
Enwiki	1	32	-4.81	-2.07	-3.52	16	-0.91	-7.85	5.27	4	-0.34	-0.02	0.02
Ruwiki	1	32	-6.93	-2.40	-3.15	16	-1.36	-10.70	5.25	4	-0.65	0.03	0.02
Zhwiki	1	32	-7.03	-2.46	-3.24	16	-1.38	-10.90	5.17	4	-0.67	0.01	0.02
		μ	**-6.26**	-2.28	-3.27		**-1.22**	-9.82	5.22		**-0.56**	0.01	0.02
		σ	0.89	0.16	0.15		0.19	1.23	0.04		0.13	0.02	0.00

Table 5.19: `u32u8` PAPI normalized difference comparison between `UTF8LUT` and `CURR`

Chapter 6

Conclusion and future work

This dissertation describes the Parabix framework. We detail the functioning of its core components and how the framework can automatically provide multi-core acceleration, guided by a program's kernel, streamset and I/O processing rate relationships.

As a framework, Parabix has improved considerably since 2017. To the programmer, the key additions are a more general notion of processing rates and attributes. Together, these allow programmers to develop more complex programs than were possible in 2017 — especially those with variable rate I/O — and provides the framework with enough information to invisibly handle streamset buffer memory management.

Schedule generation is also a major contribution of this dissertation. Although any arbitrary topological ordering of the I/O relationships could lead to a valid program, we provide a systematic approach for partitioning programs into synchronous "fixed-rate" components and leverage the existing research in synchronous dataflow literature to generate schedules that minimize pipeline overhead and buffer memory requirements. Additionally, based assumption that once a program is in its steady-state that upon invoking kernel it will exhaust some incoming streamset channel, we also introduce the concept of "partition jumping". This idea permits the pipeline to skip multiple partitions worth of processing logic when the system infers that no observable output will be generated after invoking their kernels. Schedule generation takes the possibility of these jumps into account when weighting the value of alternate options to obtain one with the least overhead.

Apart from the extended functionality, we show that when compared to prior work, the current framework is on average between 8.2 – 63.9% faster, depending on architecture and thread-count, with the best improvements occurring on newer architectures with a greater number of threads. The theoretical maximum speed-up for strict linear-pipeline parallelism is limited to ($\texttt{total single-threaded time}/\max(\texttt{total kernel time})$). Our evaluation shows that by combining the relaxed fixed-data linear-pipeline model, data-parallel execution of state-free kernels, and dataflow programming concepts, the framework exceeds this limit for two of the ten test cases, `base64` and `u8u16`, by nearly 40% and 60%, achieving a 3.5× acceleration with four threads. For six of the remaining eight cases, the framework attains

at least 83% of the limit, with half being over 90%. However, we fall short with `icgrep -colours=always`, reaching only 71%. The issue here is due to unpredictability of RE matches. When one occurs, the "matching" thread can delay other threads because of the synchronization mechanism currently used by the framework. These figures are likely to change as the studies revealed that some programs cannot easily leverage multiple threads. Specifically, `icgrep -colours=never` is effectively limited to a 2-core acceleration due to cost of its most expensive kernel. Both this case and the later studies against some of the non-Parabix programs in §5.7 highlights the importance of good program design and which programs currently fail to meet their expectations. Splitting or otherwise altering kernels whose cost dominates the run-time of the program could provide immediate performance improvements to the system.

Future work

This dissertation mentioned many avenues for future work. In our assessment against the "non-partitioned" pipeline, most programs showed a 5.4 – 13.1% improvement in single-threaded performance and a small but steady increase with higher thread counts but we observed that `icgrep -colours=never` is slower in many cases. Although the true reason(s) for this is unknown, changes in memory layout and usage is a potential culprit. Better characterization of the expected multi-threaded dataflow and an understanding of how memory layout affects performance beyond cache prefetching may help alleviate this concern. As discussed in §2.1.5, many additional avenues of thread-based acceleration are available that do not require any changes to the programs themselves. However, as Appx. D.1 indicates, the cost of the dominant kernel workload is a major limiting factor when attempting to use higher thread counts. Finally, as §5.2 recommends, adopting an n-stage execution or some form of dynamic scheduling, may help avoid the inherit drawback of our thread synchronization model for programs with probabilistically dense and sparse regions of dataflow.

Bibliography

[1] Ioannis Alagiannis, Renata Borovica, Miguel Branco, Stratos Idreos, and Anastasia Ailamaki. Nodb: efficient query execution on raw data files. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 241–252, 2012.

[2] Marco Aldinucci, Marco Danelutto, and Massimo Torquati. Fastflow tutorial. *arXiv preprint arXiv:1204.5402*, 2012.

[3] Aldeida Aleti and Irene Moser. A systematic literature review of adaptive parameter control methods for evolutionary algorithms. *ACM Computing Surveys (CSUR)*, 49(3):1–35, 2016.

[4] Gabriela Alexe, Sorin Alexe, Yves Crama, Stephan Foldes, Peter L Hammer, and Bruno Simeone. Consensus algorithms for the generation of all maximal bicliques. *Discrete Applied Mathematics*, 145(1):11–21, 2004.

[5] Gregory E Allen and Brian L Evans. Real-time sonar beamforming on workstations using process networks and posix threads. *IEEE Transactions on Signal Processing*, 48(3):921–926, 2000.

[6] John Aycock. A brief history of just-in-time. *ACM Comput. Surv.*, 35(2):97–113, June 2003.

[7] Mohammad Bakhshalipour, Mehran Shakerinava, Fatemeh Golshan, Ali Ansari, Pejman Lotfi-Karman, and Hamid Sarbazi-Azad. A survey on recent hardware data prefetching approaches with an emphasis on servers. *arXiv preprint arXiv:2009.00715*, 2020.

[8] Jonathan Curtis Beard. *Online modeling and tuning of parallel stream processing systems*. PhD thesis, 2015.

[9] Shuvra S. Bhattacharyya. *Compiling Dataflow Programs for Digital Signal Processing*. PhD thesis, EECS Department, University of California, Berkeley, Jul 1994.

[10] Shuvra S Bhattacharyya, Joe T Buck, Soonhoi Ha, and Edward A Lee. A scheduling framework for minimizing memory requirements of multirate dsp systems represented as dataflow graphs. In *Proceedings of IEEE Workshop on VLSI Signal Processing*, pages 188–196. IEEE, 1993.

[11] Shuvra S Bhattacharyya, Joseph T Buck, Soonhoi Ha, and Edward A Lee. Generating compact code from dataflow specifications of multirate signal processing algorithms.

IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, 42(3):138–150, 1995.

[12] Shuvra S Bhattacharyya and Edward A Lee. Scheduling synchronous dataflow graphs for efficient looping. *Journal of VLSI signal processing systems for signal, image and video technology*, 6(3):271–288, 1993.

[13] Shuvra S Bhattacharyya and Edward A Lee. Looped schedules for dataflow descriptions of multirate signal processing algorithms. *Formal Methods in System Design*, 5(3):183–205, 1994.

[14] Shuvra S Bhattacharyya, Praveen K Murthy, and Edward A Lee. Apgan and rpmc: Complementary heuristics for translating dsp block diagrams into efficient software implementations. *Design Automation for Embedded Systems*, 2(1):33–60, 1997.

[15] Christian Bienia and Kai Li. Characteristics of workloads using the pipeline programming model. In *International Symposium on Computer Architecture*, pages 161–171. Springer, 2010.

[16] Bruno Bodin, Luigi Nardi, Paul HJ Kelly, and Michael FP O'Boyle. Diplomat: Mapping of multi-kernel applications using a static dataflow abstraction. In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 241–250. IEEE, 2016.

[17] Jani Boutellier, Jiahao Wu, Heikki Huttunen, and Shuvra S Bhattacharyya. Prune: Dynamic and decidable dataflow for signal processing on heterogeneous platforms. *IEEE Transactions on Signal Processing*, 66(3):654–665, 2017.

[18] J Dean Brock and William B Ackerman. Scenarios: A model of non-determinate computation. In *International Colloquium on the Formalization of Programming Concepts*, pages 252–259. Springer, 1981.

[19] Adam L Buchsbaum, Howard Karloff, Claire Kenyon, Nick Reingold, and Mikkel Thorup. Opt versus load in dynamic storage allocation. *SIAM Journal on Computing*, 33(3):632–646, 2004.

[20] Joseph T. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model*. PhD thesis, EECS Department, University of California, Berkeley, 1993.

[21] Dov Bulka and David Mayhew. *Efficient C++: performance programming techniques*. Addison-Wesley Professional, 2000.

[22] Brad Calder, Chandra Krintz, Simmi John, and Todd Austin. Cache-conscious data placement. In *Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*, pages 139–149, 1998.

[23] Robert D Cameron. u8u16–a high-speed utf-8 to utf-16 transcoder using parallel bit streams. Technical report, Technical Report TR 2007-18, Simon Fraser University, Burnaby, BC, Canada, 2007.

[24] Robert D Cameron, Ehsan Amiri, Kenneth S Herdy, Dan Lin, Thomas C Shermer, and Fred P Popowich. Parallel scanning with bitstream addition: An xml case study. In *European Conference on Parallel Processing*, pages 2–13. Springer, 2011.

[25] Robert D Cameron, Kenneth S Herdy, and Dan Lin. High performance xml parsing using parallel bit stream technology. In *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, pages 222–235, 2008.

[26] Robert D Cameron and Dan Lin. Architectural support for swar text processing with parallel bit streams: the inductive doubling principle. *ACM Sigplan Notices*, 44(3):337–348, 2009.

[27] Robert D Cameron, Thomas C Shermer, Arrvindh Shriraman, Kenneth S Herdy, Dan Lin, Benjamin R Hull, and Meng Lin. Bitwise data parallelism in regular expression matching. In *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pages 139–150. IEEE, 2014.

[28] Timothy M Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 161–173. SIAM, 2010.

[29] Jiawen Chen, Michael I Gordon, William Thies, Matthias Zwicker, Kari Pulli, and Frédo Durand. A reconfigurable architecture for load-balanced rendering. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 71–80. ACM, 2005.

[30] Ron Cytron, Jeanne Ferrante, Barry K Rosen, Mark N Wegman, and F Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(4):451–490, 1991.

[31] Jan Daciuk. Comparison of construction algorithms for minimal, acyclic, deterministic, finite-state automata from sets of strings. In Jean-Marc Champarnaud and Denis Maurel, editors, *Implementation and Application of Automata*, pages 255–261, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[32] Abhishek Das, William J Dally, and Peter Mattson. Compiling for stream processing. In *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, pages 33–42, 2006.

[33] Mark Davis. Unicode regular expression guidelines. *UTR #18: Unicode Regular Expression Guidelines*, Aug 2000.

[34] Dale Lamont Denis. High-performance regular expression matching with parabix and llvm. Master's thesis, Simon Frasier University, 2014.

[35] Kristof Denolf, Marco Bekooij, Johan Cockx, Diederik Verkest, and Henk Corporaal. Exploiting the expressiveness of cyclo-static dataflow to model multimedia implementations. *EURASIP Journal on Advances in Signal Processing*, 2007(1):084078, Dec 2007.

[36] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[37] Stephen A Edwards and Olivier Tardieu. Shim: A deterministic model for heterogeneous embedded systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(8):854–867, 2006.

[38] Johan Eker, Jörn W Janneck, Edward A Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Stephen Neuendorffer, Sonia Sachs, and Yuhong Xiong. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[39] Janet Fabri. Automatic storage optimization. In *Proceedings of the 1979 SIGPLAN symposium on Compiler construction*, pages 83–91, 1979.

[40] Alireza Farshin, Amir Roozbeh, Gerald Q Maguire Jr, and Dejan Kostić. Make the most out of last level cache in intel processors. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–17, 2019.

[41] Antony A Faustini. An operational semantics for pure dataflow. In *International Colloquium on Automata, Languages, and Programming*, pages 212–224. Springer, 1982.

[42] G. R. Gao, R. Govindarajan, and P. Panangaden. Well-behaved dataflow programs for dsp computation. In *[Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 561–564 vol.5, March 1992.

[43] Chang Ge, Yinan Li, Eric Eilebrecht, Badrish Chandramouli, and Donald Kossmann. Speculative distributed csv data parsing for big data analytics. In *Proceedings of the 2019 International Conference on Management of Data*, pages 883–899, 2019.

[44] Marc Geilen and Twan Basten. Kahn process networks and a reactive extension. In *Handbook of signal processing systems*, pages 967–1006. Springer, 2010.

[45] Jordan Gergov. Approximation algorithms for dynamic storage allocation. In *European Symposium on Algorithms*, pages 52–61. Springer, 1996.

[46] Jordan Gergov. Algorithms for compile-time memory optimization. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 907–908, 1999.

[47] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

[48] Michael I Gordon and Saman Amarasinghe. *Compiler techniques for scalable performance of stream programs on multicore architectures*. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering . . . , 2010.

[49] Thierry Goubier, Renaud Sirdey, Stéphane Louise, and Vincent David. σc: A programming model and language for embedded manycores. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 385–394. Springer, 2011.

[50] Ruirui Gu, Jörn W Janneck, Mickaël Raulet, and Shuvra S Bhattacharyya. Exploiting statically schedulable regions in dataflow programs. *Journal of Signal Processing Systems*, 63(1):129–142, 2011.

[51] Ahmad Hassanat, Khalid Almohammadi, Esra' Alkafaween, Eman Abunawas, Awni Hammouri, and VB Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12):390, 2019.

[52] Kenneth Stuart Herdy. *s2k: A parallel language for streaming text extraction and transformations*. PhD thesis, Simon Frasier University, 2015.

[53] Stephen K How. Code generation for multirate dsp systems in gabriel. Master's thesis, 1990.

[54] Chia-Jui Hsu and Shuvra S Bhattacharyya. Cycle-breaking techniques for scheduling synchronous dataflow graphs. Technical report, 2007.

[55] Hua Huang. Idisa+: A portable model for high performance simd programming. Master's thesis, Applied Science: School of Computing Science, 2011.

[56] Ibrahim Hur and Calvin Lin. Memory prefetching using adaptive stream detection. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 397–408. IEEE, 2006.

[57] Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. Here are my data files. here are my queries. where are my results? *Proceedings of 5th Biennial Conference on Innovative Data Systems Research*, 2011. SYSTEMS.

[58] R Intel. Intel 64 and ia-32 architectures optimization reference manual. *Intel Corporation*, June 2021.

[59] Gilles Kahn. The semantics of a simple language for parallel programming. *Information processing*, 74:471–475, 1974.

[60] Gilles Kahn and David MacQueen. Coroutines and networks of parallel processes. 1976.

[61] Richard M Karp and Rayamond E Miller. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM Journal on Applied Mathematics*, 14(6):1390–1411, 1966.

[62] Manos Karpathiotakis, Ioannis Alagiannis, and Anastasia Ailamaki. Fast queries over heterogeneous data through engine customization. *Proceedings of the VLDB Endowment*, 9(12):972–983, 2016.

[63] Manos Karpathiotakis, Ioannis Alagiannis, Thomas Heinis, Miguel Branco, and Anastasia Ailamaki. Just-in-time data virtualization: Lightweight data management with vida. In *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR)*, number CONF, 2015.

[64] Manos Karpathiotakis, Miguel Branco, Ioannis Alagiannis, and Anastasia Ailamaki. Adaptive query processing on raw data. *Proceedings of the VLDB Endowment*, 7(12):1119–1130, 2014.

[65] Yong-Hyuk Kim, Yourim Yoon, and Zong Woo Geem. A comparison study of harmony search and genetic algorithm for the max-cut problem. *Swarm and evolutionary computation*, 44:130–135, 2019.

[66] S. Kohli. Cache aware scheduling for synchronous dataflow programs. Master's thesis, Feb 2004.

[67] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[68] Geoff Langdale and Daniel Lemire. Parsing gigabytes of json per second. *The VLDB Journal*, 28(6):941–960, 2019.

[69] Chris Lattner. LLVM: An Infrastructure for Multi-Stage Optimization. Master's thesis, Computer Science Dept., University of Illinois at Urbana-Champaign, Urbana, IL, Dec 2002.

[70] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, Sep. 1987.

[71] Edward Ashford Lee and David G Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Transactions on computers*, 100(1):24–35, 1987.

[72] Kang Seok Lee, Zong Woo Geem, Sang-ho Lee, and Kyu-woong Bae. The harmony search heuristic algorithm for discrete structural optimization. *Engineering Optimization*, 37(7):663–684, 2005.

[73] Daniel Lemire and Wojciech Muła. Transcoding billions of unicode characters per second with simd instructions. *Software: Practice and Experience*, 2021.

[74] Yinan Li, Nikos R Katsipoulakis, Badrish Chandramouli, Jonathan Goldstein, and Donald Kossmann. Mison: a fast json parser for data analytics. *Proceedings of the VLDB Endowment*, 10(10):1118–1129, 2017.

[75] Dan Lin. *Multidimensional Parallelization for Streaming Text Processing Applications Based on Parabix Framework*. PhD thesis, Simon Frasier University, 2017.

[76] Dan Lin, Nigel Medforth, Kenneth S Herdy, Arrvindh Shriraman, and Rob Cameron. Parabix: Boosting the efficiency of text processing on commodity processors. In *IEEE International Symposium on High-Performance Comp Architecture*, pages 1–12. IEEE, 2012.

[77] Meng Lin. Systematic support of parallel bit streams in llvm. Master's thesis, Simon Frasier University, 2014.

[78] Aristeidis Mastoras. *Efficient Execution of Linear Pipelines*. PhD thesis, ETH Zurich, 2019.

[79] Aristeidis Mastoras and Thomas R Gross. Unifying fixed code and fixed data mapping of load-imbalanced pipelined loops. *ACM SIGPLAN Notices*, 51(8):1–2, 2016.

[80] Michael McCool, James Reinders, and Arch Robison. *Structured parallel programming: patterns for efficient computation*. Elsevier, 2012.

[81] Nigel Woodland Medforth. icxml: Accelerating xerces-c 3.1. 1 using the parabix framework. 2013.

[82] Mohammad Alaul Haque Monil, Seyong Lee, Jeffrey S Vetter, and Allen D Malony. Understanding the impact of memory access patterns in intel processors. In *2020 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, pages 52–61. IEEE, 2020.

[83] Tobias Mühlbauer, Wolf Rödiger, Robert Seilbeck, Angelika Reiser, Alfons Kemper, and Thomas Neumann. Instant loading for main memory databases. *Proceedings of the VLDB Endowment*, 6(14):1702–1713, 2013.

[84] Wojciech Muła and Daniel Lemire. Faster base64 encoding and decoding using avx2 instructions. *ACM Transactions on the Web (TWEB)*, 12(3):1–26, 2018.

[85] Wojciech Muła and Daniel Lemire. Base64 encoding and decoding at almost the speed of a memory copy. *Software: Practice and Experience*, 50(2):89–97, 2020.

[86] Praveen K. Murthy. *Scheduling Techniques for Synchronous and Multidimensional Synchronous Dataflow*. PhD thesis, University of California, Berkeley, 1996.

[87] Praveen K Murthy and Shuvra S Bhattacharyya. Shared memory implementations of synchronous dataflow specifications. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition 2000 (Cat. No. PR00537)*, pages 404–410. IEEE, 2000.

[88] Praveen K Murthy and Shuvra S Bhattacharyya. Shared buffer implementations of signal processing systems using lifetime analysis techniques. *IEEE transactions on computer-aided design of integrated circuits and systems*, 20(2):177–198, 2001.

[89] Praveen K Murthy and Shuvra S Bhattacharyya. Buffer merging—a powerful technique for reducing memory requirements of synchronous dataflow specifications. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 9(2):212–237, 2004.

[90] Praveen K Murthy, Shuvra S Bhattacharyya, and Edward A Lee. Joint minimization of code and data for synchronous dataflow programs. *Formal Methods in System Design*, 11(1):41–70, 1997.

[91] Craig Mustard, Fabian Ruffy, Anny Gakhokidze, Ivan Beschastnikh, and Alexandra Fedorova. Jumpgate: In-network processing as a service for data analytics. In *11th {USENIX} Workshop on Hot Topics in Cloud Computing (HotCloud 19)*, 2019.

[92] André Nieuwland, Jeffrey Kang, Om Prakash Gangwal, Ramanathan Sethuraman, Natalino Busá, Kees Goossens, Rafael Peset Llopis, and Paul Lippens. C-heap: A heterogeneous multi-processor architecture template and scalable and flexible protocol for the design of embedded signal processing systems. *Design Automation for Embedded Systems*, 7(3):233–270, 2002.

[93] Shoumik Palkar, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Filter before you parse: Faster analytics on raw data with sparser. *Proceedings of the VLDB Endowment*, 11(11):1576–1589, 2018.

[94] Thomas M Parks. *Bounded scheduling of process networks*. PhD thesis, Berkeley Univ., California, 1995.

[95] James Lyle Peterson. *Petri net theory and the modeling of systems*. Prentice-hall, 1981.

[96] Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit te Leuven, 2003.

[97] Robert Clay Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[98] Junqiao Qiu, Lin Jiang, and Zhijia Zhao. Challenging sequential bitstream processing via principled bitwise speculation. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 607–621, 2020.

[99] S. Ritz, M. Pankert, V. Zivojinovic, and H. Meyr. Optimum vectorization of scalable synchronous dataflow graphs. In *Proceedings of International Conference on Application Specific Array Processors (ASAP '93)*, pages 285–296, 1993.

[100] Sebastian Ritz, Matthias Pankert, and Heinrich Meyr. High level software synthesis for signal processing systems. In *Proceedings of the international conference on application specific array processors*, pages 679–680, 1992.

[101] Sebastian Ritz, Markus Willems, and Heinrich Meyr. Scheduling for optimum data memory compaction in block diagram oriented software synthesis. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 4, pages 2651–2654. IEEE, 1995.

[102] Aditya Rohan, Biswabandan Panda, and Prakhar Agarwal. Reverse engineering the stream prefetcher for profit. pages 682–687, 2020.

[103] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. *ACM SIGARCH Computer Architecture News*, 41(1):305–316, 2013.

[104] Jan Hendrik Schönherr. *Coscheduling in the multicore era: the art of doing things simultaneously*. Doctoral thesis, Technische Universität Berlin, Berlin, 2019.

[105] Lars Schor, Iuliana Bacivarov, Devendra Rai, Hoeseok Yang, Shin-Haeng Kang, and Lothar Thiele. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems*, pages 71–80. ACM, 2012.

[106] Arthur Stoutchinin. *A Dataflow Framework For Developing Flexible Embedded Accelerators A Computer Vision Case Study.* PhD thesis, alma, 2019.

[107] Martin Sulzmann and Kenny Zhuo Ming Lu. Posix regular expression parsing with derivatives. In *International Symposium on Functional and Logic Programming*, pages 203–220. Springer, 2014.

[108] Dan Terpstra, Heike Jagode, Haihang You, and Jack Dongarra. Collecting performance data with papi-c. pages 157–173, 2010.

[109] Bill Thies, Michal Karczmarek, and Saman Amarasinghe. Streamit: A language for streaming applications. 2001.

[110] William Frederick Thies. *Language and compiler support for stream programs*. PhD thesis, Massachusetts Institute of Technology, 2009.

[111] Brian CH Turton. Extending quine-mccluskey for exclusive-or logic synthesis. *IEEE Transactions on Education*, 39(1):81–85, 1996.

[112] Henry S Warren. *Hacker's delight*. Pearson Education, 2013.

[113] Maarten H Wiggers, Marco JG Bekooij, and Gerard JM Smit. Buffer capacity computation for throughput constrained streaming applications with data-dependent inter-task communication. In *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 183–194. IEEE, 2008.

[114] Paul R Wilson, Mark S Johnstone, Michael Neely, and David Boles. Dynamic storage allocation: A survey and critical review. In *International Workshop on Memory Management*, pages 1–116. Springer, 1995.

[115] Dong Xue. Advanced techniques for bounded and unbounded repetition in parabix regular expression search. Master's thesis, Simon Frasier University, 2017.

[116] Shiyang Yang. Validation of xml document based on parallel bit stream technology. Master's thesis, Simon Frasier University, 2013.

[117] Xuejun Yang, Li Wang, Jingling Xue, Yu Deng, and Ying Zhang. Comparability graph coloring for optimizing utilization of stream register files in stream processors. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 111–120, 2009.

[118] Xuejun Yang, Li Wang, Jingling Xue, and Qingbo Wu. Comparability graph coloring for optimizing utilization of software-managed stream register files for stream processors. *TACO*, 9(1):5–1, 2012.

[119] Tonghua Zhang and Zong Woo Geem. Review of harmony search with respect to algorithm structure. *Swarm and Evolutionary Computation*, 48:31–43, 2019.

[120] Eckart Zitzler, Jürgen Teich, and Shuvra S Bhattacharyya. Optimized software synthesis for dsp using randomization techniques. In *Tech. Rep. 75, Institute TIK, ETH Zurich, Gloriastrasse 35, CH-8092*. Citeseer, 1999.

[121] Eckart Zitzler, Jürgen Teich, and SS Bhattclcharyya. Evolutionary algorithms for the synthesis of embedded software. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(4):452–455, 2000.

Appendix A

Two modifications to Yang et al.'s offline DSA algorithm

In §3.2.3, we discussed Yang et al. offline DSA algorithm tailored for dataflow systems. Their algorithm made a critical assumption that most outputs were immediately consumed by the subsequent node in the graph [117, 118]. Such streamsets could be encoded directly within the comparability graph G_0. For partition-local subgraphs, this is true — but the majority of inter-partition streamsets have more than one consumer, immediately violating Yang et al.'s assumption. Such streamsets are represented in the induced subgraph $\mathcal{G}(\mathcal{V}_2)$ constructed from the original streamset interval graph G_I after removing any streamset nodes that are represented in G_0. The process of generating G_I is discussed in §3.2.2.

Because of the authors' assumption, they conject that $\mathcal{G}(\mathcal{V}_2)$ is almost always a forest [117] and when it is not, a taking a maximal spanning forest of $\mathcal{G}(\mathcal{V}_2)$ suffices [118]. This approach is problematic because a spanning forest arbitrarily destroys interval relationships in $\mathcal{G}(\mathcal{V}_2)$, leading to an under-approximation of the actual memory requirements of a schedule. However, they do not actually need a forest. Yang et al. wanted something that was trivially recognizable as a (connected) prime comparability graph and since all forests are bipartite graphs, which must be prime comparability graphs, it sufficed for their purpose.

Closely related to the spanning-tree problem is the notion of a max-cut. Given a graph $G = (V, E)$, the **weighted max-cut** problem asks what set $S \subseteq V$ maximizes total weight of the edges between S and $V \setminus S$. Defined formally, it asks given edge weights w_{ij} for each edge $(i, j) \in E$, what set S satisfies Eq. A.1:

$$\max \sum_{i<j} w_{ij} \frac{1 - x_i \cdot x_j}{2}, \text{ where } x_k = \begin{cases} 1 & v_k \in S \\ -1 & v_k \in V \setminus S \end{cases} \tag{A.1}$$

Computing a max-cut for $\mathcal{G}(\mathcal{V}_2)$ could preserve more information but there are two significant problems:

1. weighted interval graph colouring is a vertex-weighted problem — not an edge-weighted one. To calculate edge weights, we use Eq. A.2. Our intuition is that we want to prioritize

placing two moderate weight streamsets into separate partitions over a heavy and light pair. When cutting an edge, we lose interval relationship in the graph. Our assumption is that we will only under-approximate the required weight by the lower of the two streamset nodes.

$$\frac{(X+Y)^3}{4(X^2+Y^2)} \tag{A.2}$$

We base our use of Eq. A.2 on the following inequalities:

a) $\frac{(X+Y)^3}{4(X^2+Y^2)} > \frac{((X+1)+(Y-1))^3}{4((X+1)^2+(Y-1)^2)}$ True when $X \geq Y$

b) $\frac{(X+Y)^3}{4(X^2+Y^2)} < \frac{((X+1)+Y)^3}{4((X+1)^2+Y^2)}$ True for positive integers X and Y

2. the running time of Yang et al.'s offline DSA algorithm is 2^{N+1} where N is the number of connected components (CCs) in MAX-CUT($\mathcal{G}(\mathcal{V}_2)$). To the best of our knowledge, no existing heuristic for max-cuts explicitly preserves CCs. Preliminary tests using `icgrep 'a' -colours=always` found greedy heuristics increased N from 4 to 16. Depending on the size of $\mathcal{G}(\mathcal{V}_2)$, we may be able to permit a max-cut that decomposes $\mathcal{G}(\mathcal{V}_2)$ into a small number of CCs but must do so sparingly. To that end, we redefine our max-cut optimization goal as Eq. A.3, where n is the number of CCs in the cut decomposition and k is some constant indicating the maximum number of CCs desired for any decomposition of $\mathcal{G}(\mathcal{V}_2)$.

$$\max \sqrt[\gamma]{\sum_{i<j} w_{ij} \frac{1 - x_i \cdot x_j}{2}}, \text{ where } \gamma = \begin{cases} n > k & (n-k+1) \\ otherwise & 1 \end{cases} \tag{A.3}$$

For the first modification to Yang et al.'s algorithm, we adapted Kim et al.'s max-cut harmony search algorithm[1] [65] and reevaluated their approach with our optimization function (Eq. A.3) in §A.1. The second modification to Yang et al.'s algorithm is significantly more straightforward. Neither [117] nor [118] address what to do about the loss of information incurred by taking a spanning tree/max-cut.

The comparability graph method only determines the chromatic number and does not return a colouring. However, given a disconnected graph G, we can colour each CC with unique sets of colours to obtain a sub-optimal colouring of G. Thus to obtain a safe over-approximation of the true colouring of G_I, we add chromatic numbers of G_I and that of the residual interval graph G_R (i.e., a graph with non-isolated vertices and weights of $\mathcal{G}(\mathcal{V}_2)$) but only the edge relationships that were not in the max-cut). To obtain the chromatic number of G_R, we

[1]Harmony search is a evolutionary algorithm inspired by musical harmonization [72] and consists of three components: harmony memory, harmony memory consideration rate (HMCR) and pitch adjustment rate (PAR). Each candidate of the harmony memory is a m-length vector composed of $[-1, 1]$ values. At each step of the algorithm, a new candidate is chosen wherein each i^{th} value of the new candidate selects the i^{th} value of some j^{th} vector in the memory with HMCR probability or is randomly generated $[-1, 1]$ value otherwise. For a given harmony memory size of n, only the top n candidates are kept in the memory after each iteration. Pitch adjustment is randomly modifies each i^{th} value of the new candidate by $\pm\epsilon$ with PAR probability after assignment. Kim et al. limit their algorithm to only consider discrete Boolean values and forgo pitch adjustment. [65]

recursively execute our modified DSA algorithm on G_R. This, however, is an obvious topic for improvement.

A.1 Evaluation

Initial experimentation in §4.3 found that over 90% of the time spend scheduling complex programs was directly attributed to our max-cut and recursive modifications to the DSA algorithm when the harmony search was run to convergence. The following study was performed to determine whether the modifications were viable or whether some other approach should be considered. Our conclusion found a balance between quality and speed that appears satisfactory in terms of performance and recursion depth as a fitness-cost function.

A.1.1 Experiment 1.

In Kim et al.'s evaluation of their algorithm, they found a HMCR of 0.997 (which we adopted for our initial study) provided the best mean average cut size [65]. They compared their harmony search approach against a standard bitstring-based EA. In our experiment, we reproduced both of their algorithms but generate 6 sets of graphs of varying edge insertion probability rates of 5% to 50%, which we believe covers our range of expected use cases. Each set consisted of 10 graphs of 100 nodes — representing a very large but not unreasonable number of streamsets. Each node was randomly assigned a weight of $[4, 64] \times 1024$, which are typical allocation sizes in a `-SegmentSize=4096` program. We present 5 configurations in Figure A.1, where each configuration is presented as a line in the graph depicting the average maximum cut-size at each round/time interval.

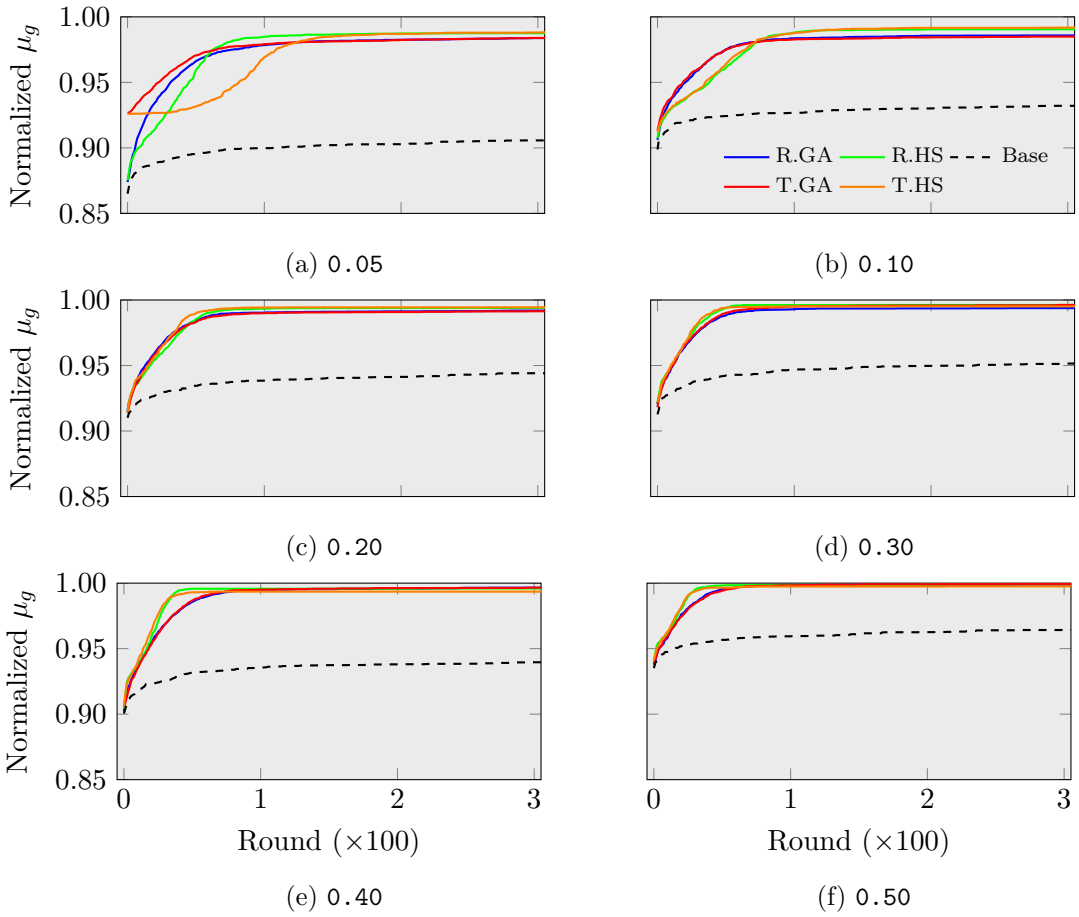


Figure A.1: Max-cut Evolutionary Algorithm comparison

For the `Base` configuration, we generate 100 candidates every round, retaining the best for the next round; each `Base` candidate is a random assignments of $x_i \in \{-1, 1\}$ values. The fitness of all candidates is determined by Eq. A.3. The two configurations with a "`R.`" prefix are initialized with 20 random candidates, similar to the `Base` configuration. Under the assumption we could converge to an optimal solution faster with better initial candidates, the configurations with a "`T.`" prefix are initially seeded by variants of Prim's [97] and Kruskel's [67] spanning-tree algorithms. Edge selection for each variant is a maximal weighted-random choice. Since every tree T is a bipartite graph whose partitions contain the vertices in the even or odd levels of T, the levels of T dictate whether a vertex v_i is in subset S or $V \setminus S$ of the max-cut and the subset v_i belongs to dictates the sign of x_i. Both spanning-tree algorithms add 10 candidates each to the initial set. The suffix `GA` and `HS` indicate whether the genetic algorithm or harmony search was used to converge upon a solution, respectively.

To compute each data line in Figure A.1, we ran 10 trials for each configuration of each graph and logged the weight of the current best-known cut after each round. To normalize the cut-sizes, we divided each data point for a graph G by the maximum cut-size across all trials for G. At each data point in Figure A.1, we plot the geometric mean of the normalized values across all 10 graphs with the same edge insertion probability. Each trial was run for 1,000

rounds but only the first 300 are shown in Figure A.1 since no further improvements were made by any approach. To minimize any potential bias introduced by the initial populations, each of the 10 pairs of otherwise independent *X*.GA and *X*.HS trials used the same initial candidates. Similarly, because HS technically generates one new candidate per iteration, we normalize the notion of round between trial pairs by only considering each n_i^{th} best result of a HS trace where n_i is the total number of candidates generated after the i^{th} round of the paired GA trace.

Immediately obvious from Figure A.1 is that the Base configuration is the worst-performing one but the y-axis scale is somewhat misleading since even the initial random solutions were $\geq 85\%$ of maximum weight. Interestingly, the spanning-tree approaches, while initially better in sparser graphs are essentially indistinguishable from random assignments in graphs with $\geq 20\%$ edge density. Just as Kim et al.'s evaluation concluded, the harmonic search does outperform a genetic algorithm for this problem but whereas they reported a $\approx 19\%$ improvement, we see only a negligible difference in our randomly generated graphs after 150 rounds. This discrepancy is potentially a byproduct of the change between the standard max-cut optimization criteria and Eq. A.3 or definition of round but further investigation exceeds the scope of this thesis.

A.1.2 Experiment 2.

Because we intend to use our modified version of Yang et al.'s offline DSA approach as a *fitness function* for a genetic algorithm, efficiency is critical. Early experimentation in §4.3 found that over 90% of the time spend scheduling complex programs was directly attributed to our max-cut and recursive modifications to the DSA algorithm. Kim et al. recommended settings were intended for finding an optimal max-cut solution to a single problem, not to repeatedly execute it as a fitness function. The following study was performed to determine whether the modifications were viable or whether some other approach should be considered. Our conclusion found a balance between quality and speed that appears satisfactory in terms of performance and recursion depth as a fitness-cost function.

For our problem, we must consider efficiency from two points of view: 1) the time we require to compute the max-cut and 2) the quality of the max-cut itself since we recursively execute the DSA algorithm on the set of edges not contained within the cut set. Specifically, we want to determine how many iterations are required before we can be confident that the best known solution is within 5% of the optimal solution and what settings will lead us to such a solution faster than others with a 95% confidence interval. We call such cut-sets a 95%-cut. Recall that the harmony search has three parameters: harmonic memory (i.e., population size), HMCR and pitch. Since Kim et al.'s algorithm relied on discrete Boolean variables, they ignored pitch [65]. Thus here we focus on the first two settings.

Based on prior works, Kim et al. used a harmonic memory size of 10 and determined that a HMCR of 0.997 provided the best results for the max-cut problem [65]. In general, Lee et al. recommend sizes of between 10 and 50 and a HMCR of $0.70 - 0.95$, based on empirical evidence [72]. However, in both of these cases the problems were to converge on an optimal solution — not a near-optimal one quickly.

Configuration combinations	
Nodes	10,20,30,40,50,60,70,80,90,100
Edge density	0.05,0.10,0.20,0.30,0.40,0.50
HM	**10**,15,20,25,30,40,50,75,100,150
HMCR	0.7,0.8,0.85,**0.9**,0.925,0.95,0.997

Table A.1: Harmony search evaluation configurations

To evaluate Kim et al.'s parameters in the context of our problem, for each combination of # of nodes and edge densities, we generate a set of 50 non-bipartite graphs of the appropriate shape and evaluate each graph using all pairs of harmony memory and HMCR settings. Each graph was tested 10 times using each HM/HMCR pairing and every trial was limited to 100,000 iterations. For each trial, we determined the first iteration that achieved a 95%-cut w.r.t. the best-known cut for the graph or assigned a score of 200,000 for any trial that failed to converge on a 95%-cut by the 100,000-iteration cutoff. For the 500 trials of each combination, we computed the geometric mean of their 95%-cut iteration score then determined the upper limit of where the true geomean lies at the 95% confidence interval.

		# of Nodes									
		10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Edge density	**0.05**	10	10	10	10	10	10	10	10	10	10
	0.1	10	10	10	10	10	10	15	15	15	20
	0.2	10	10	10	10	10	10	25	15	25	25
	0.3	10	10	10	10	10	15	20	15	20	30
	0.4	10	10	10	10	10	10	10	10	10	15
	0.5	10	10	10	10	10	10	10	10	10	10

Table A.2: Best harmonic memory (population size)

		# of Nodes									
		10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Edge density	**0.05**	.700	.850	.800	.900	.950	.925	.950	.950	.950	.950
	0.1	.700	.800	.925	.925	.950	.925	.950	.950	.950	.950
	0.2	.850	.900	.925	.950	.925	.900	.950	.950	.950	.950
	0.3	.850	.925	.900	.850	.850	.900	.950	.900	.925	.925
	0.4	.800	.800	.850	.900	.850	.850	.900	.925	.850	.925
	0.5	.950	.850	.950	.925	.925	.900	.900	.900	.950	.925

Table A.3: Best HMCR

		Harmonic Memory (Population size)									
		10	**15**	**20**	**25**	**30**	**40**	**50**	**75**	**100**	**150**
	.700	32490.3	45340.2	52922.4	65399.2	77001.0	97668.7	117628.6	165862.4	215714.2	307142.7
	.800	535.5	928.7	1533.0	2233.5	3232.3	5384.6	8981.8	20994.0	35163.9	74049.7
	.850	48.2	97.6	183.4	282.3	446.0	855.0	1445.5	3820.1	7618.7	20494.2
	.900	**3.4**	10.1	28.1	56.8	98.8	229.6	410.9	1175.0	2431.8	6674.9
	.925	4.6	7.4	20.3	35.3	63.6	147.2	266.6	776.2	1653.1	4524.8
	.950	6.8	6.4	14.1	23.9	47.7	99.3	188.3	541.1	1156.7	3182.5
	.997	1109.5	368.7	173.4	143.3	149.6	149.0	209.9	457.0	857.0	2148.5

Table A.4: Residual sum of squares of ratio between rank-value of $(i,j)^{\texttt{th}}$ and best configuration

$$RSS_{i,j} = \sum_{\substack{\mathcal{G}\in\text{Graph-sets},\\ \mathcal{C}\in\text{Configs}}} \left(\frac{\texttt{Rank-value}\left(\mathcal{C}_{i,j}(\mathcal{G})\right)}{\texttt{Rank-value}\left(\mathcal{C}_{best}(\mathcal{G})\right)} - 1 \right)^2 \tag{A.4}$$

For each graph set $\mathcal{G}$, we rank the configurations $\mathcal{C}$ according to *pop_size* + *upper_limit* (representing the total number of calls to the cut evaluation function) and identified the the best (i.e., minimal) configuration for each graph set. In Tbl. A.2 and Tbl. A.3, we list the HM and HMCR setting of the best configuration. Although a regression model could be calculated for them, Tbl. A.4 shows this is unnecessary. When reviewing the data, we observed that the difference between the top-3 rankings was vanishingly small in many cases. To identify whether we could apply a single static configuration setting to this problem, we computed the residual sum of squares between the ratios of each configuration of a graph set (shown in Eq. A.4) and determined that a HM of 10 and a HMCR of 0.9 was — of our tested configurations — a near-optimal configuration for every graph set. Thus for the remainder of this section, we consider this our static configuration setting.

		# of Nodes									
		10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
Edge density	**0.05**	2	3	145	795	949	925	1045	1333	1731	2422
	0.1	2	137	214	305	626	766	1275	1494	2048	2672
	0.2	11	106	259	379	854	1371	1969	2087	1991	3189
	0.3	17	63	232	724	1225	1635	2240	3530	3196	8283
	0.4	11	149	390	583	1120	1606	2842	2686	2428	5277
	0.5	12	117	192	479	1189	966	2286	2672	2168	5382
	0.75	11	114	200	270	491	692	1010	1558	995	1482
	1.0	19	55	209	248	225	208	427	642	408	348

Table A.5: Expected # of iterations before obtaining 95%-cut at 99% CL using FIXED(0.9) HMCR model

The next stage of this experiment is to develop a regression model for our chosen configuration that reliably indicates the running time of the max cut algorithm. In Tbl. A.5, we present our findings for the expected number of iterations required to to obtain a 95%-cut at a 95% confidence level. To produce Tbl. A.5, we reuse the graphs from the prior stage but add 50 new graphs for each edge density 0.75 and 1.00 setting. Like before, 10 trials were performed on each graph but increase our harmony search cutoff to 1,000,000 iterations. Even so, we observed that 7.6% of trials failed to locate a 95%-cut by the cutoff. Initially, we attempted to incorporate that data into the entries of Tbl. A.5 by assigning a score of 1,000,000 but doing so lead to regression functions with extremely negative residuals for many trials. Instead, we omitted those entries in Tbl. A.5 to obtain a tighter prediction. As expected, small graphs tend to quickly locate a 95%-cut. Interestingly, this also appears to be true of denser graphs. This may be a byproduct of the Pythagorean weight function rather than a general rule but bears further investigation.

$$\begin{aligned} f(n,m) &= \alpha \cdot n^{\beta} + 142.9282 \\ \alpha &= \text{POLY4}(\gamma, 233.0784, -151.8367, 36.1286, -3.7582, 0.1529) \\ \beta &= \text{POLY4}(\gamma, 260.3499, -208.0714, 21.0946, 6.8772, 2.0361) \\ \gamma &= {}^{m}/_{(n\cdot(n-1))} \\ \text{POLY4}(x,a,b,c,d,e) &= a \cdot x^4 + b \cdot x^3 + c \cdot x^2 + d \cdot x + e \end{aligned} \tag{A.5}$$

$$\textbf{Adjusted } \mathbf{R^2 = 0.9116}$$

Based on Tbl. A.5, we developed the regression model detailed in Eq. A.5, where $n = |V|$ and $m = |E|$ for a given graph $G = (V, E)$. Figure A.2 depicts the residual graph, where

the shade indicates the geomean of the residuals between the number of iterations passed before first finding a 95%-cut and the expected number of iterations as predicted by Eq. A.5 of the trials that achieved a 95%-cut by the 1,000,000 iteration cutoff.

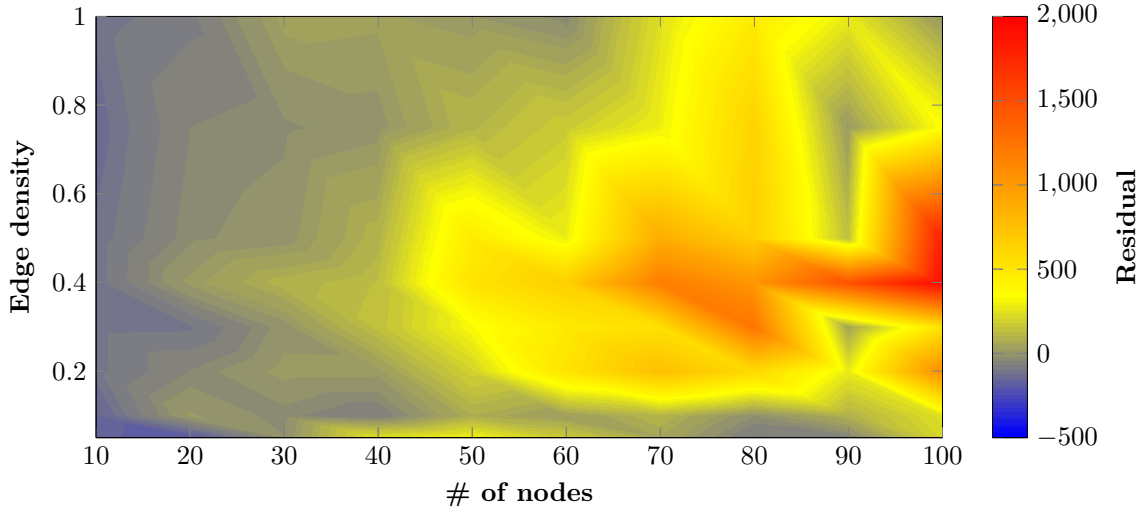


Figure A.2: Contour graph of residual between actual data and Eq. A.5 using FIXED(0.9) HMCR model

Edge density	# of Nodes									
	10	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
0.05	-156	-197	-18	251	313	179	58	-76	-55	210
0.1	-152	15	-32	-66	77	26	101	-20	88	249
0.2	-108	-37	21	19	200	521	760	592	277	1084
0.3	-107	-104	-6	138	327	483	548	1243	41	535
0.4	-112	18	92	130	524	658	1187	1032	1473	1950
0.5	-114	-25	-5	94	477	287	883	661	136	1817
0.75	-132	-39	-25	-3	102	176	292	632	15	351
1.0	-112	-77	40	35	12	-31	238	485	274	5

Table A.6: Geomean of residuals between actual data and Eq. A.5 using FIXED(0.9) HMCR model

Let ℓ be the result of Eq. A.5. Immediately evident from Figure A.2 is that Eq. A.5 poorly matches the data but does provide some assurance that setting the cutoff for any subsequent experiments to $2 \times \ell + 1000$ is reasonable. Although we could continue to find a better regression formula or investigate whether a better fixed consideration rate exists, we instead look to dynamic HMCR models. Dynamic models vary the HMCR rate over time according to some mathematical function. The use of dynamic HMCR models is a well-known concept [119] and a similar concept have been shown to be beneficial in genetic algorithms [3, 51]. With the simplifying assumption that a harmonic memory size of 10 would be ideal for all max-cut harmony search problems, we evaluate a normal and absolute cosinusoidal HMCR function in Eq. A.6 and Eq. A.7, where $0 \leq L < H \leq 1$ bounds the range, f is cosine frequency and k indicates we are on the $k^{\texttt{th}}$ of ℓ iteration the max-cut algorithm. The intuition behind both functions is that since we initially start with a random harmonic memory, we want to identify the best qualities of those random solutions before exploring new ones and afterwards periodically transition between exploration and

harmonizing states. The difference is that Eq. A.6 spends roughly an equal amount of time in both states, Eq. A.7 slightly prioritizes harmonization.

$$\textsc{Cos}(L, H, P, x) = \left(\frac{H-L}{2}\right) \cdot \texttt{cos}\left(\frac{2\pi}{P}x\right) + \left(\frac{H+L}{2}\right) \tag{A.6}$$

$$\textsc{AbsCos}(L, H, P, x) = (H-L) \cdot \left|\texttt{cos}\left(\frac{\pi}{P}x\right)\right| + L \tag{A.7}$$

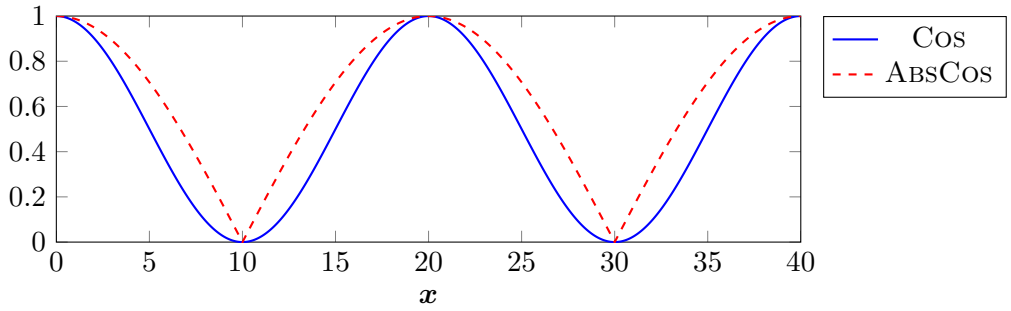


Figure A.3: Example plot of Eq. A.6 and Eq. A.7

We reevaluated all 4000 graphs, again performing 10 trials on each graph bounding each trial to $2 \times \ell + 1000$, to explore the impact of the tuning parameters for Eq. A.6 and Eq. A.7 on the expected run-time of the max-cut algorithm. Our initial assessment of these models can be seen in Tbl. A.7. For each individual graph, we compute expected # of iterations before first finding a 95%-cut at the 99% CL. We computed a residual sum of squares for a chosen sets of parameters, using the minimal expected iteration as our theoretical estimate and normalizing the result using Eq. A.8 to prevent larger graph configurations from dominating the result. Although Tbl. A.7 shows that both Eq. A.6 and Eq. A.7 can out perform the $\textsc{Fixed}(0.900)$ model with some sets of parameters, $\textsc{Cos}(0.8, 1.0, 20)$ HMCR model appears to be the best choice of the parameter combinations tested. Interestingly, the second best combination is $\textsc{AbsCos}(0.7, 1.0, 20)$. This gives additional weight to the idea that a period of 20 (for a harmony memory size of 10) may be near optimal but whether these observations can be used to improve the HMCR model warrants future study.

$$RSS_{i,j} = \sum_{\substack{\mathcal{G} \in \text{All graphs}, \\ \mathcal{P} \in \text{Parameters}}} \left(\frac{\texttt{Upper-limit}\left(\mathcal{P}_{i,j}(\mathcal{G})\right) - \texttt{Upper-limit}\left(\mathcal{P}_{min}(\mathcal{G})\right.}{\texttt{Upper-limit}\left(\mathcal{P}_{max}(\mathcal{G})\right) - \texttt{Upper-limit}\left(\mathcal{P}_{min}(\mathcal{G})\right.} \right)^2 \tag{A.8}$$

		Fixed								
0.900		371.8								
		Period length (P)								
		2	4	6	8	10	20	30	40	50
Cos Range (L–H)	0.0–0.9	388.6	842.1	1048.3	1075.2	1085.7	1080.1	1105.5	1088.1	1090.0
	0.0–1.0	1440.4	1065.7	627.2	436.8	384.4	371.0	356.6	380.0	386.0
	0.5–0.9	230.3	397.3	459.4	451.1	470.8	467.7	455.5	460.3	457.0
	0.5–1.0	1110.5	406.9	208.9	198.5	187.3	174.3	181.6	185.3	183.3
	0.7–0.9	184.8	249.4	251.1	264.8	244.3	268.7	274.1	263.8	250.0
	0.7–1.0	656.8	193.1	153.3	147.4	156.1	150.5	143.2	140.3	148.9
	0.8–0.9	158.7	174.6	177.2	177.7	174.2	175.4	179.2	181.1	184.4
	0.8–1.0	335.5	152.5	139.6	138.1	140.2	**129.3**	134.8	139.4	138.3
	0.9–1.0	176.9	160.3	158.1	163.5	149.4	155.3	156.1	150.5	143.4
		60	70	80	90	100	125	150	175	200
Cos Range (L–H)	0.0–0.9	1097.5	1077.5	1094.7	1109.8	1105.8	1090.7	1109.7	1082.8	1099.8
	0.0–1.0	369.0	379.7	366.2	380.6	374.1	377.8	376.1	382.5	368.3
	0.5–0.9	471.8	464.6	475.0	468.3	475.7	458.8	464.2	467.4	446.2
	0.5–1.0	176.2	180.1	190.0	188.4	191.4	192.7	193.4	189.3	192.2
	0.7–0.9	262.8	261.7	261.6	259.5	255.9	257.0	249.2	265.4	266.7
	0.7–1.0	141.2	157.2	147.6	154.3	142.5	138.2	152.9	153.8	151.0
	0.8–0.9	182.6	174.3	178.6	183.7	175.2	187.7	174.7	184.0	185.7
	0.8–1.0	151.0	139.7	157.0	142.2	146.0	145.2	133.6	147.4	149.7
	0.9–1.0	149.7	153.4	147.5	154.7	155.3	151.2	141.4	150.0	142.4
		Period length (P)								
		2	4	6	8	10	20	30	40	50
AbsCos Range (L–H)	0.0–0.9	391.1	580.4	640.9	643.1	646.0	638.0	633.7	646.1	650.2
	0.0–1.0	1408.1	596.6	265.9	232.9	225.8	217.9	220.0	232.0	214.5
	0.5–0.9	223.3	280.6	307.5	304.8	303.6	309.2	287.8	298.4	302.0
	0.5–1.0	1107.0	202.6	146.1	146.2	137.9	160.1	149.1	158.7	152.1
	0.7–0.9	176.5	194.5	205.1	198.1	187.7	200.9	204.7	196.7	199.9
	0.7–1.0	643.6	152.0	145.9	142.0	154.3	**130.7**	143.4	157.3	144.4
	0.8–0.9	162.0	167.3	160.2	164.4	153.1	152.2	153.5	172.0	168.0
	0.8–1.0	331.2	147.7	143.1	142.7	151.8	147.4	145.3	157.0	142.5
	0.9–1.0	184.3	183.9	189.5	176.9	176.8	166.3	174.1	168.3	162.2
		60	70	80	90	100	125	150	175	200
AbsCos Range (L–H)	0.0–0.9	635.9	626.9	643.0	638.7	628.6	638.0	640.5	623.0	628.2
	0.0–1.0	219.8	226.7	215.7	204.9	220.4	225.0	226.1	215.6	234.7
	0.5–0.9	290.5	296.6	305.3	317.9	310.0	281.5	300.8	301.0	300.3
	0.5–1.0	164.7	154.0	158.8	147.0	150.5	161.0	166.8	153.0	150.8
	0.7–0.9	202.3	202.0	198.8	192.8	200.6	198.1	199.5	200.5	204.6
	0.7–1.0	136.8	147.2	155.1	136.8	144.8	139.0	134.7	146.5	148.9
	0.8–0.9	156.5	164.2	158.4	146.1	156.2	155.6	154.9	154.4	156.3
	0.8–1.0	158.3	151.4	142.8	149.9	147.9	152.8	147.9	140.8	143.6
	0.9–1.0	177.7	164.8	174.7	158.3	178.7	177.6	167.2	177.0	181.0

Table A.7: Residual sum of squares of ratio between expected and best # of iterations by HMCR model

With $\textsc{Cos}(0.8, 1.0, 20)$ selected as our preferred model, we repeat the process used to generate Tbl. A.5 and calculate the expected number of iterations required to to obtain a 95%-cut at a 99% confidence level. Tbl. A.8 presents our results and provide a regression equation for it in Eq. A.9. We display the contour graph of the residual geomeans in Figure A.4. Although Figure A.4 fits slightly better than Figure A.2 indicates $\textsc{Fixed}(0.900)$, it is important to note that Eq. A.9 is both asymptotically smaller than Eq. A.5 but also requires $\sim 51.2\%$ fewer iterations than predicted by Eq. A.5 ($\sigma_g = 1.201$).

Edge density	# of Nodes									
	10	20	30	40	50	60	70	80	90	100
0.05	1	4	82	325	498	661	806	921	1095	1324
0.1	2	107	193	292	447	592	870	981	1178	1400
0.2	12	99	201	319	456	600	797	804	1007	1308
0.3	15	71	203	329	509	550	710	927	930	1338
0.4	14	119	229	276	472	640	770	857	628	1264
0.5	13	70	190	290	384	428	757	752	908	1066
0.75	11	96	176	194	297	386	414	623	712	767
1.0	22	50	219	132	188	215	360	540	393	331

Table A.8: Expected number of iterations before obtaining 95%-cut at 99% CL using $\textsc{Cos}(0.8, 1.0, 20)$ HMCR model

$$g(n, m) = 0.07930 \cdot n^2 + 7.63712 \cdot n - 0.19735 \cdot m - 80.59364 \tag{A.9}$$

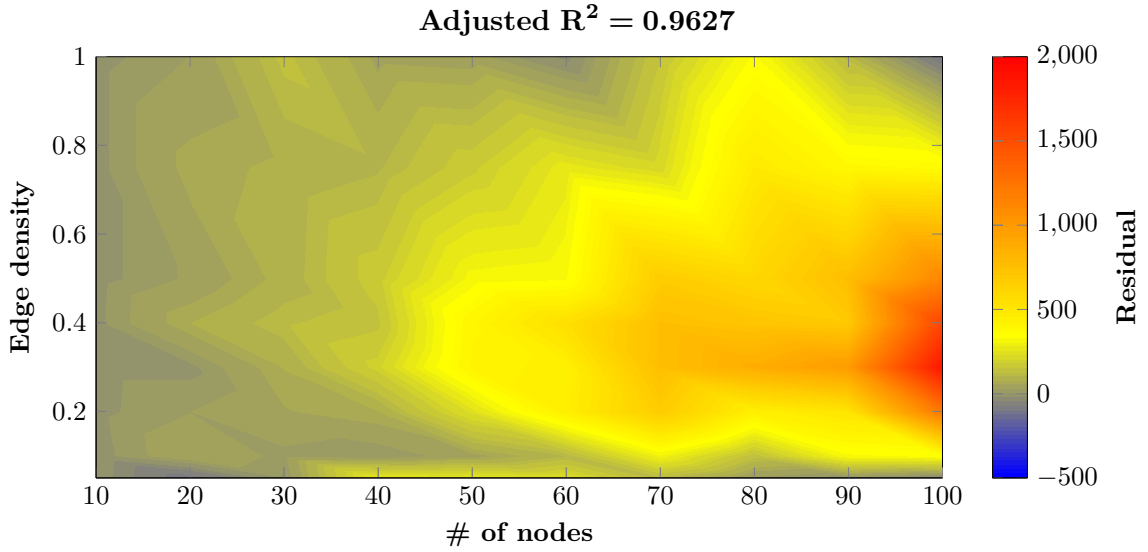


Figure A.4: Contour graph of residual geomean between actual data and Eq. A.9 using $\textsc{Cos}(0.8, 1.0, 20)$ HMCR model

Edge density	# of Nodes									
	10	20	30	40	50	60	70	80	90	100
0.05	-4	-91	7	274	267	240	117	22	-6	-42
0.1	-2	55	22	11	46	108	302	139	325	344
0.2	3	29	56	77	243	449	681	444	514	1229
0.3	2	2	80	197	417	443	762	877	986	1859
0.4	2	78	123	149	407	549	759	721	692	1553
0.5	0	34	89	176	315	328	680	607	763	1128
0.75	0	69	84	104	179	269	219	483	392	375
1.0	0	30	137	51	44	-29	156	341	116	-99

Table A.9: Geomean of residuals between actual data and Eq. A.9 using $\textsc{Cos}(0.8, 1.0, 20)$ HMCR model

As a final comparison between expected cut-sizes of $\textsc{Cos}(0.8, 1.0, 20)$ and $\textsc{Fixed}(0.9)$ at the iteration indicated by Eq. A.5 and Eq. A.9, we plotted the frequency distribution of both models. Recall that to formulate Eq. A.5, we ran each trial for 1,000,000 iterations, which biased the results of Tbl. A.5 (and consequently Eq. A.5) towards larger numbers. Although from Tbl. A.7 we expect that $\textsc{Cos}(0.8, 1.0, 20)$ will outperform $\textsc{Fixed}(0.9)$ at Eq. A.9, we cannot conclude from Tbl. A.7 by how much. When considering only the probability of

obtaining a 95%-cut (i.e., $\mathrm{P}(X \geq 0.95)$), it is clear that $\textsc{Cos}(0.8, 1.0, 20)$ skews slightly more to the right than $\textsc{Fixed}(0.9)$ at either iteration point but the difference is negligible at Eq. A.5. However, it does appears our assumption about setting the harmonic memory size to 10 was reasonable. Although $\mathrm{P}(X \geq 0.95)$ is significantly worse for both models at Eq. A.9 than Eq. A.5, $\textsc{Cos}(0.8, 1.0, 20)$ does skew reasonably towards the 95%-cut goal. Thus $\textsc{Cos}(0.8, 1.0, 20)$ at Eq. A.9 appears to be an acceptable performance compromise.

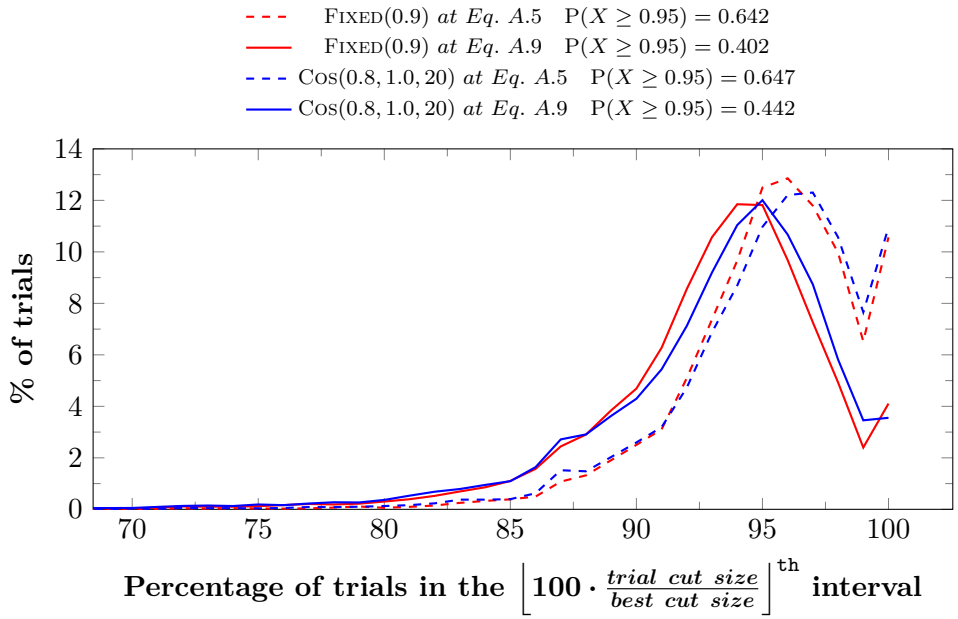


Figure A.5: Frequency distribution comparison between $\textsc{Fixed}(0.9)$ and $\textsc{Cos}(0.8, 1.0, 20)$

Appendix B

Closest-match EA repair function

In §4.3.2, we introduced the problem of obtaining the closest match for an EA-candidate from an arbitrary permutation of chromosomes when given a DAWG whose transition labels define the space of viable individuals. Although its relatively straightforward to seed a candidate with the permutation, finding the closest candidate is significantly harder. Because this function is just one component of a large multi-traversal EA, ideally we want a function that we can control how much time is spent searching for a closest match but also one that converge upon a near-optimal solution after a finite number of steps.

For this problem we combine the ant colony optimization (ACO) heuristic with the Kendall tau distance function. The ACO heuristic was originally used to solve the travelling salesman problem [36]. With it, artificial ants would traverse a graph G using both edge weights and pheromones to guide their path selection with the goal of locating the shortest path through G. ACOs are typically divided into two repeating stages: exploration (i.e., solution generation) and pheromone update. During the exploration phase, each ant performs a weighted-random walk through G wherein the probability of moving from vertex x to y is:

$$p_{xy} = \frac{(\tau_{xy}^{\alpha})(\eta_{xy}^{\beta})}{\sum_{z \in allowed_x}(\tau_{xz}^{\alpha})(\eta_{xz}^{\beta})} \tag{B.1}$$

Here $allowed_u$ is the set of unvisited vertices adjacent from u, τ_{uv} and η_{uv} are the accumulated pheromones and static weights on each (u, v) edge, respectively. The exponents α and β are constants in which α influence the importance of posteriori knowledge gained by repeated walks and β any static knowledge provided by the designer. Since our problem is purely an exploration-based one, we fix $\eta_{uv}^{\beta} = 1.0$ and initialize all values of τ_{uv} to 5.0 and arbitrarily set $\tau_{min} = 0.1$ and $\tau_{max} = 10,000.0$.

During the pheromone update stage, we evaluate each source-to-sink walk through the DAWG G and alter the pheromone level of every edge, reinforcing the probability of choosing attractive paths and reducing it on others. Unfortunately, the $\Delta\tau_{uv} = Q/L$ pheromone update stage discussed by Dorigo et al. [36] cannot be applied here since we do not have a meaningful constant to substitute for Q nor is tour length L related to the problem. Our approach for updating τ_{uv} is as follows:

Assume that C and K is the EA candidate and the sequence of kernel labels observed during a chosen walk. We use Eq. B.2 to compute an inversion cost that indicates the total disagreement between our lists. Kendall tau distance function utilized here is a ranking metric that counts the number of disagreements between two ordered lists and is equivalent to the number of bubble-sort swaps necessary to transform one list to the other; this can be computed in $\mathcal{O}(|C|\sqrt{\log|C|})$ time [28].

$$\textsc{InversionCost}(C, K) = \frac{2 \cdot \textsc{KendallTauDist}(C, K)}{|C|(|C| - 1) + 1} < 1 \tag{B.2}$$

Let $\mathcal{I}_{current}$ and $\mathcal{I}_{best}$ be the costs found for the current and best-known (lowest-cost) traversals. Initially, $\mathcal{I}_{best} = 1$. For every (u, v) edge on a chosen path we update τ_{uv} by:

$$\tau_{uv}(t+1) = \max\left(\min\left(\tau_{uv}(t) + \frac{\mathcal{I}_{best} - \mathcal{I}_{current}}{|\mathcal{I}_{best} - \mathcal{I}_{current}| + k}, \tau_{max}\right), \tau_{min}\right) \tag{B.3}$$

Because the interval in which Eq. B.3 changes τ_{uv} by is $[-1, 1]$, we forgo the standard pheromone evaporation/decay process of most ACOs. Early experiments showed by even with small values of ρ, evaporation would either not alter the outcome or (as ρ increases) resulted in homogeneous values for τ_{uv}. The k-value in Eq. B.3 influences the importance of small differences between $\mathcal{I}_{current}$ and $\mathcal{I}_{best}$.

Since every traversal through the DAWG G is a valid replacement, the ACO algorithm can safely terminate after a single traversal but since this function is intended to find the closest matching replacement, more traversals are likely needed. However, because this algorithm is intended to be a repair function for §4.3.2, we want it to provide us with a good result in the fewest number of traversals. We can achieve this by by tuning k and α for our system. To investigate, we assess 4 programs here, detailed in Tbl. B.1. The columns $|V|$, $|E|$ and $|C|$ report the geometric mean (μ_g) and geo. standard deviation (σ_g) of the number of vertices and edges in G, and the length of each candidate, corresponding to the number of selected edges in every path through G.

Program	|V|	|E|	|C|
`gb18030`	64.66 (1.36)	79.35 (1.42)	41.0 (1.0)
`icgrep 'a' -colours=always`	59.97 (1.08)	59.94 (1.09)	57.0 (1.0)
`u32u8`	29.73 (1.30)	35.52 (1.33)	19.0 (1.0)
`ztfhash -d`	60.96 (1.13)	64.33 (1.19)	52.0 (1.0)

Table B.1: Selected programs

We varied the values of $k = \{0.001, 0.01, 0.1, 0.2, 0.3, \ldots, 1.0\}$, and the values of $\alpha = \{0.0, 0.5, 1.0, \ldots 6.0\}$. We generated 10 independent trials for each program, selecting $100,000$ runs of each configuration, executing each for $10,000$ traversals of G. We computed the geomean of the number of traversals required until first observing the best computed solution for each ACO candidate. A best solution K is the one with a lowest $\textsc{InversionCost}(C, K)$ for a unique pair of DAWG and candidate C. The geomeans of the top 100 ranked configurations for `icgrep`, `u32u8` and `ztfhash` were all under 25 but `gb18030` varied between 109.7 and

143.7. Based on this, we selected five potential traversal cutoff points, 25, 50, 100, 150 and 200, and computed the residual sum of squares comparing the difference between the solution found at an traversal cutoff and the best computed solution for each candidate, presented in Tbl. B.2 – B.6. The top five configurations are bolded for visual ease.

We found that better solutions were found more quickly with k-values of 0.01 and 0.001 up to traversal 150 and only at traversal 200 did any of the top five configurations include $k = 0.1$. α-values of 1.5 or more were almost always favoured for those k-value groups and 0.0 (i.e., a random exploration) was never included in the top five configurations — but $k = 0.01$ and $\alpha = 6.0$ is. Selecting $k = 0.01$ and $\alpha = 6.0$ as our configuration, the next question is how well does it perform compared to the best configuration and how many traversals are sufficient to reliably obtain a good solution?

		α												
		0.0	**0.5**	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**	**5.5**	**6.0**
k	**0.001**	9.44	10.85	13.01	**5.61**	**7.74**	**6.50**	8.55	8.86	8.13	**7.00**	10.31	10.06	15.28
	0.010	9.26	9.13	7.92	10.40	10.07	8.86	8.38	10.88	11.63	9.73	11.48	9.47	**6.43**
	0.100	8.95	9.14	10.08	12.92	13.24	12.05	10.40	10.38	11.83	14.04	19.74	13.13	13.35
	0.200	15.87	9.37	9.57	10.84	16.97	12.56	18.42	12.80	19.48	14.37	15.35	11.99	12.44
	0.300	12.04	15.08	10.86	10.10	15.12	10.31	9.76	9.31	12.87	12.07	13.16	16.03	12.08
	0.400	11.71	13.56	11.43	12.99	12.13	12.91	12.89	11.32	11.91	16.59	14.59	11.45	12.99
	0.500	10.41	11.92	12.46	12.73	11.72	13.53	12.32	14.82	11.72	11.79	21.14	13.01	11.59
	0.600	9.84	16.01	11.15	12.82	12.10	15.65	13.98	18.71	16.05	11.25	15.66	9.99	12.45
	0.700	13.09	10.71	10.52	12.79	13.62	11.20	13.48	13.13	12.29	13.79	15.17	13.44	13.44
	0.800	11.55	14.49	12.39	16.18	15.06	15.79	10.24	12.28	11.68	16.14	10.55	14.20	10.98
	0.900	12.48	13.15	10.04	14.26	11.31	14.45	11.72	11.30	11.12	11.59	14.84	12.13	15.85
	1.000	10.08	14.19	10.83	10.28	20.64	12.60	13.27	14.00	12.27	12.18	13.85	14.20	13.22

Table B.2: Residual sum of squares ($\times\mathbf{10^{-2}}$) of achieved vs. best solution at **t=25**

		α												
		0.0	**0.5**	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**	**5.5**	**6.0**
k	**0.001**	3.58	3.13	2.91	**1.43**	**1.42**	1.53	1.58	1.50	**1.41**	**1.19**	1.97	1.81	2.62
	0.010	3.17	2.54	1.96	2.69	1.82	1.81	1.71	2.10	2.70	2.01	2.25	2.13	**1.25**
	0.100	2.94	2.71	3.40	5.14	5.41	4.41	3.59	3.30	4.48	5.34	8.48	5.02	4.84
	0.200	5.41	3.15	3.57	3.56	6.66	4.36	7.44	5.10	8.02	5.56	6.77	3.91	5.00
	0.300	4.21	5.50	3.57	3.44	5.43	3.00	2.95	2.90	5.08	4.20	5.00	6.79	4.34
	0.400	3.89	4.74	4.48	5.34	4.60	4.90	4.63	3.86	4.04	6.49	5.40	4.55	4.65
	0.500	3.56	4.39	4.10	4.54	4.14	5.56	4.50	5.50	4.44	4.02	8.72	4.65	4.64
	0.600	3.21	6.52	4.06	5.01	4.46	5.62	5.16	7.25	6.40	3.95	6.17	3.22	4.74
	0.700	4.64	3.63	3.77	4.41	4.84	4.01	5.26	4.70	3.73	5.25	5.77	5.69	5.41
	0.800	4.37	5.68	4.70	6.07	5.54	6.46	3.42	4.35	3.73	6.06	3.77	5.59	3.88
	0.900	4.21	4.80	3.48	5.15	4.12	5.72	4.48	4.14	4.13	4.31	5.98	4.65	5.83
	1.000	3.60	5.00	3.61	3.47	8.00	4.19	5.08	4.83	4.48	4.20	4.92	5.33	4.75

Table B.3: Residual sum of squares ($\times\mathbf{10^{-2}}$) of achieved vs. best solution at **t=50**

		α												
		0.0	**0.5**	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**	**5.5**	**6.0**
k	**0.001**	10.98	7.81	7.64	5.24	5.05	5.14	5.25	5.32	4.54	4.27	5.91	5.30	6.04
	0.010	10.44	6.24	5.16	5.23	4.80	4.06	**3.38**	4.13	4.31	**3.39**	**3.63**	**3.23**	**2.59**
	0.100	10.96	9.22	10.95	18.21	17.41	14.74	9.74	8.84	13.51	13.24	20.51	12.30	11.97
	0.200	17.86	10.22	11.23	10.43	22.52	13.60	22.36	13.51	25.18	16.61	21.20	10.22	13.82
	0.300	12.75	17.74	13.03	11.19	18.38	10.07	9.81	9.21	17.19	13.54	14.74	26.12	12.51
	0.400	12.04	14.66	13.90	17.48	15.54	17.10	15.20	10.90	13.65	20.41	17.13	15.00	15.10
	0.500	11.49	12.94	13.59	13.95	14.60	18.24	13.40	16.44	15.29	12.09	29.33	15.31	13.17
	0.600	11.83	21.37	13.98	16.01	13.87	23.13	16.30	21.39	20.46	12.45	21.36	11.85	13.04
	0.700	15.02	11.58	12.06	14.44	15.04	12.55	17.82	15.91	13.14	18.78	19.29	17.64	18.02
	0.800	13.45	19.18	15.39	18.08	16.57	22.02	10.62	15.28	14.16	19.19	12.24	17.12	12.83
	0.900	14.70	13.50	12.21	17.20	14.21	19.44	14.14	13.67	12.78	14.70	19.06	14.07	18.98
	1.000	11.46	17.71	12.50	11.64	22.65	14.31	18.21	17.45	15.19	14.18	15.64	17.53	14.70

Table B.4: Residual sum of squares ($\times\mathbf{10^{-3}}$) of achieved vs. best solution at **t=100**

		α												
		0.0	**0.5**	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**	**5.5**	**6.0**
k	**0.001**	6.49	4.33	4.31	3.08	3.08	3.01	3.25	3.24	2.94	2.84	3.62	3.62	3.55
	0.010	6.42	3.42	2.61	2.49	2.42	**2.11**	**2.06**	2.46	2.28	2.15	**2.09**	**2.10**	**1.76**
	0.100	5.89	4.14	3.99	5.94	5.27	4.20	2.95	2.96	3.80	3.89	6.87	4.03	2.56
	0.200	10.18	5.63	6.09	5.51	11.82	5.54	8.37	5.16	6.87	7.05	7.13	4.42	4.88
	0.300	7.20	8.95	6.87	6.13	9.05	5.76	5.30	4.70	7.68	4.84	5.64	9.50	5.74
	0.400	7.00	8.86	7.78	9.85	8.13	8.36	7.25	5.53	6.15	9.09	7.46	6.56	6.89
	0.500	5.62	7.68	8.32	7.62	7.71	9.51	7.46	7.85	6.49	6.23	10.69	6.91	6.12
	0.600	6.60	10.48	8.27	9.38	7.74	11.71	8.74	10.65	10.46	6.90	10.16	5.79	7.46
	0.700	9.11	6.60	7.41	7.81	7.68	7.10	8.73	8.59	6.51	10.51	8.99	8.04	8.71
	0.800	7.71	10.16	8.47	10.32	10.12	11.25	6.42	8.32	7.41	11.62	6.82	8.68	7.49
	0.900	8.74	8.07	7.17	9.87	7.49	10.66	8.46	8.35	6.81	7.30	9.87	7.61	10.24
	1.000	6.71	9.79	7.55	7.03	11.87	7.51	9.77	8.73	7.19	7.36	9.09	8.74	8.27

Table B.5: Residual sum of squares ($\times\mathbf{10^{-3}}$) of achieved vs. best solution at **t=150**

		α												
		0.0	**0.5**	**1.0**	**1.5**	**2.0**	**2.5**	**3.0**	**3.5**	**4.0**	**4.5**	**5.0**	**5.5**	**6.0**
k	**0.001**	4.30	3.00	3.01	2.34	2.38	2.15	2.41	2.39	2.27	2.08	2.81	2.68	2.71
	0.010	4.39	2.40	1.88	1.76	1.66	1.48	1.61	1.88	1.82	1.65	**1.48**	1.50	**1.40**
	0.100	3.94	2.78	2.26	2.56	2.07	1.89	1.51	**1.47**	1.92	**1.30**	2.44	1.90	**1.07**
	0.200	6.29	3.34	3.45	2.82	5.84	2.95	3.51	2.55	2.88	3.09	3.46	1.97	2.25
	0.300	4.43	5.89	3.87	3.74	5.00	3.75	2.98	2.51	3.10	2.72	2.69	3.38	2.58
	0.400	4.55	5.54	4.51	6.20	4.55	4.73	3.81	3.35	3.46	4.24	3.95	3.55	3.15
	0.500	3.97	5.11	5.97	4.93	4.45	5.71	4.79	4.19	3.53	3.39	5.39	3.98	3.79
	0.600	4.62	6.91	5.18	5.78	5.31	6.74	5.61	6.65	5.50	4.15	5.41	3.68	4.92
	0.700	5.88	4.65	5.00	4.94	5.17	4.88	5.61	5.64	4.11	5.12	4.91	4.88	4.86
	0.800	5.02	6.51	5.53	6.44	6.48	6.68	4.03	5.70	4.79	6.50	4.27	4.94	4.72
	0.900	5.79	5.54	4.96	6.08	5.03	6.31	5.54	4.89	4.22	4.18	6.39	4.65	5.56
	1.000	4.70	5.98	4.66	4.88	7.55	4.81	6.36	5.45	4.61	4.54	5.39	5.70	5.02

Table B.6: Residual sum of squares ($\times\mathbf{10^{-3}}$) of achieved vs. best solution at **t=200**

In Figure B.1, we plot a comparison between the residual sum of squares between the selected $k = 0.01$ and $\alpha = 6.0$ configuration and the best configuration for every traversal. Although not shown here, there is a significant difference with fewer than 10 traversals, by 50 traversals the difference between these models is $\approx 6 \times 10^{-4}$. Although these plots do not converge until traversal 7820, at 100 traversals the difference between the obtained solution and the best known one is 0.0025855. Consequently, we select 100 as our cutoff point.

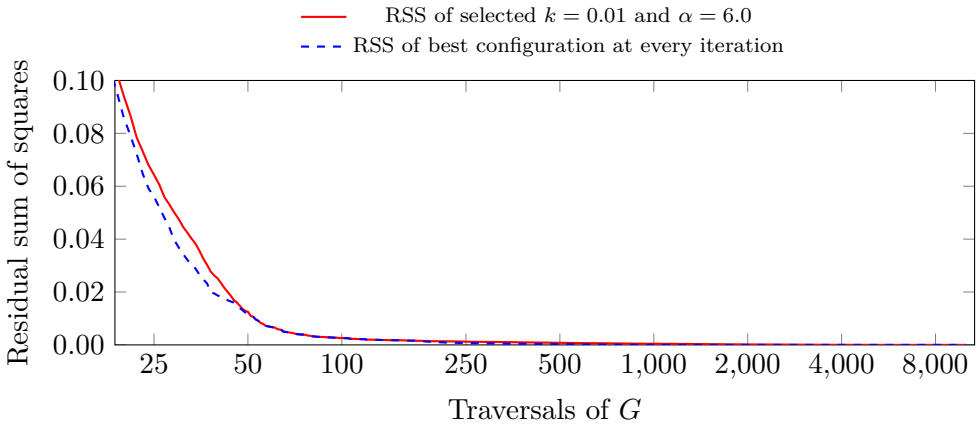


Figure B.1: Comparison between selected and best configuration

Appendix C

Program dataflow graphs

(a) Kernel (b) StreamSet

Figure C.1: Pipeline graph nodes

This appendix presents diagrams of the pipelines for the programs assessed in Ch. 5 on `AVX2`. Rounded rectangles represent kernels and (standard) rectangles represent streamsets. The directed edges between kernels and streamsets depict I/O channels where the labels beside each indicate the processing rate (relative to the stride length) and the friendly "binding" name associated with the kernel port. An example of a kernel node is shown in Figure C.1a. *number* represents the position of the kernel in the linear pipeline, *name* is the name that the framework uses to identify whether a cached instance of the kernel exists, and the *range* dictates the minimum and maximum number of strides per segment. To ensure every partition behaves like a consistent SDF network, the actual number of strides per logical invocation will always be a multiple of the lower bound.

Streamsets are more complicated: given the example in Figure C.1b, a indicate the internal streamset id of a node and b is a type code for the streamset buffer. Type codes prefixes include, `S`, `D`, `E`, indicating whether the streamset is a static, dynamic, or external buffer. An `L` marks the buffer as a linear buffer; all other buffers are circular. For a full description of these buffer types, see §2.1.3. c states the number of streams in the streamset, d the bit-width of each token, and e indicates the static or initial capacity of the buffer divided by the SIMD register size.

Partitions are indicated by solid dashed red boxes around a group of kernels and streamsets objects. Streamsets with blue boxes are thread-local streamsets, meaning they are both partition-local and do not have any segment-to-segment dependencies such as `LookAhead` or `LookBehind` I/O attributes. Any streamset that fails any of the thread-local requirements are marked with a black box. Any partition jump to a partition that is not the successive partition in the schedule is indicated with a dotted red line.

C.1 base64

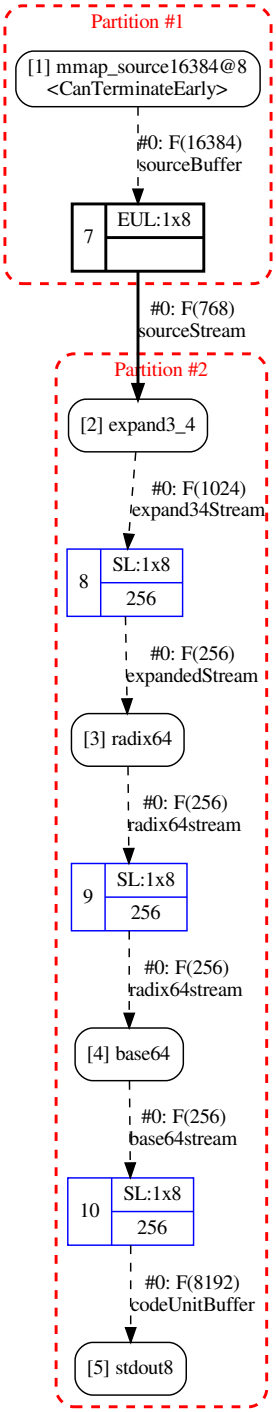


Figure C.2: Pipeline graph for base64

C.2 csv2json

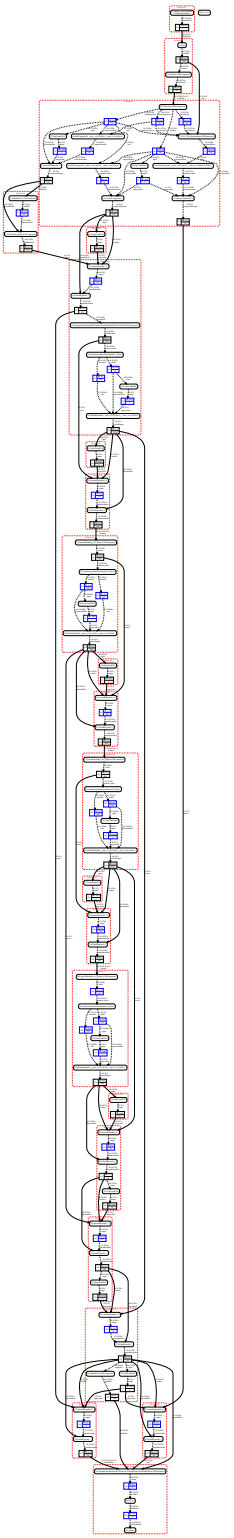


Figure C.3: Pipeline graph for csv2json

C.3 editd

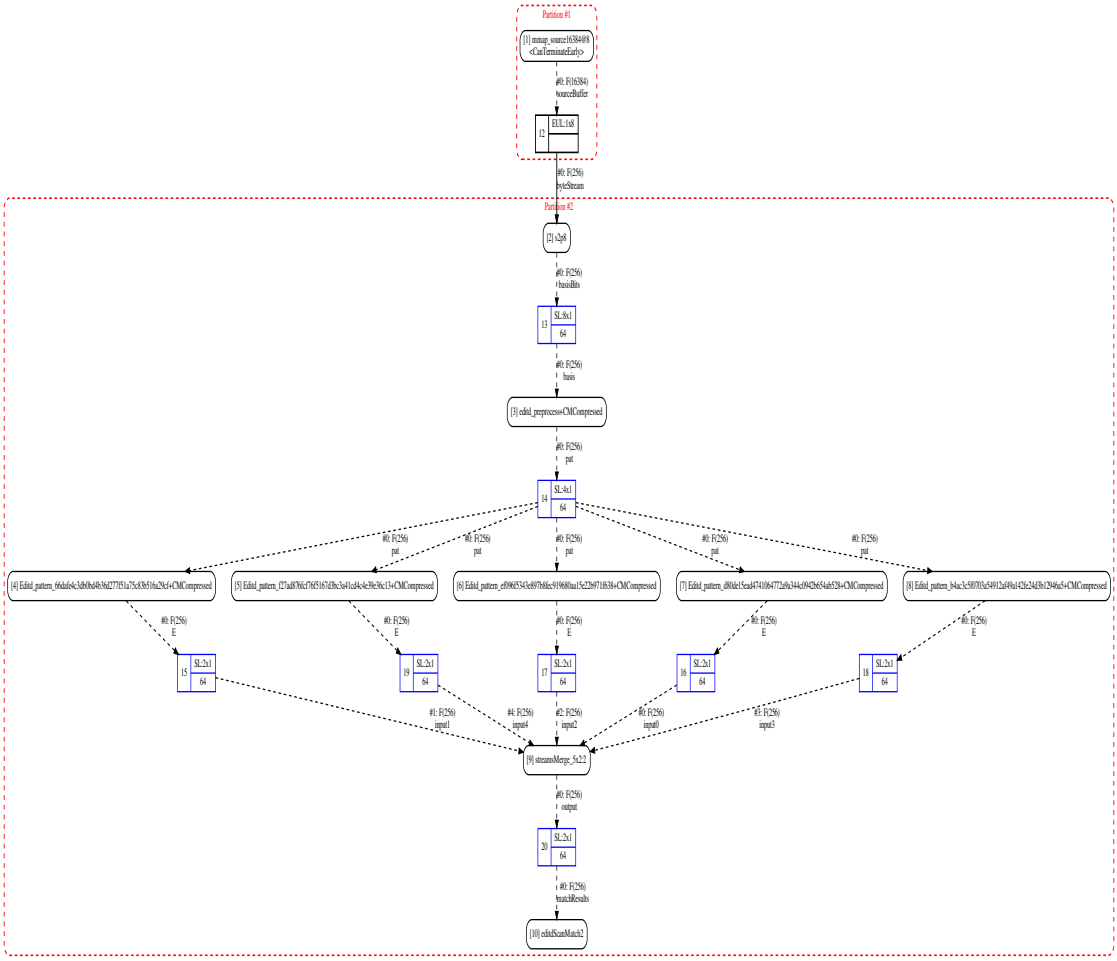


Figure C.4: Sample pipeline graph for multi-editd with 5 pattern kernels

C.4 gb18030

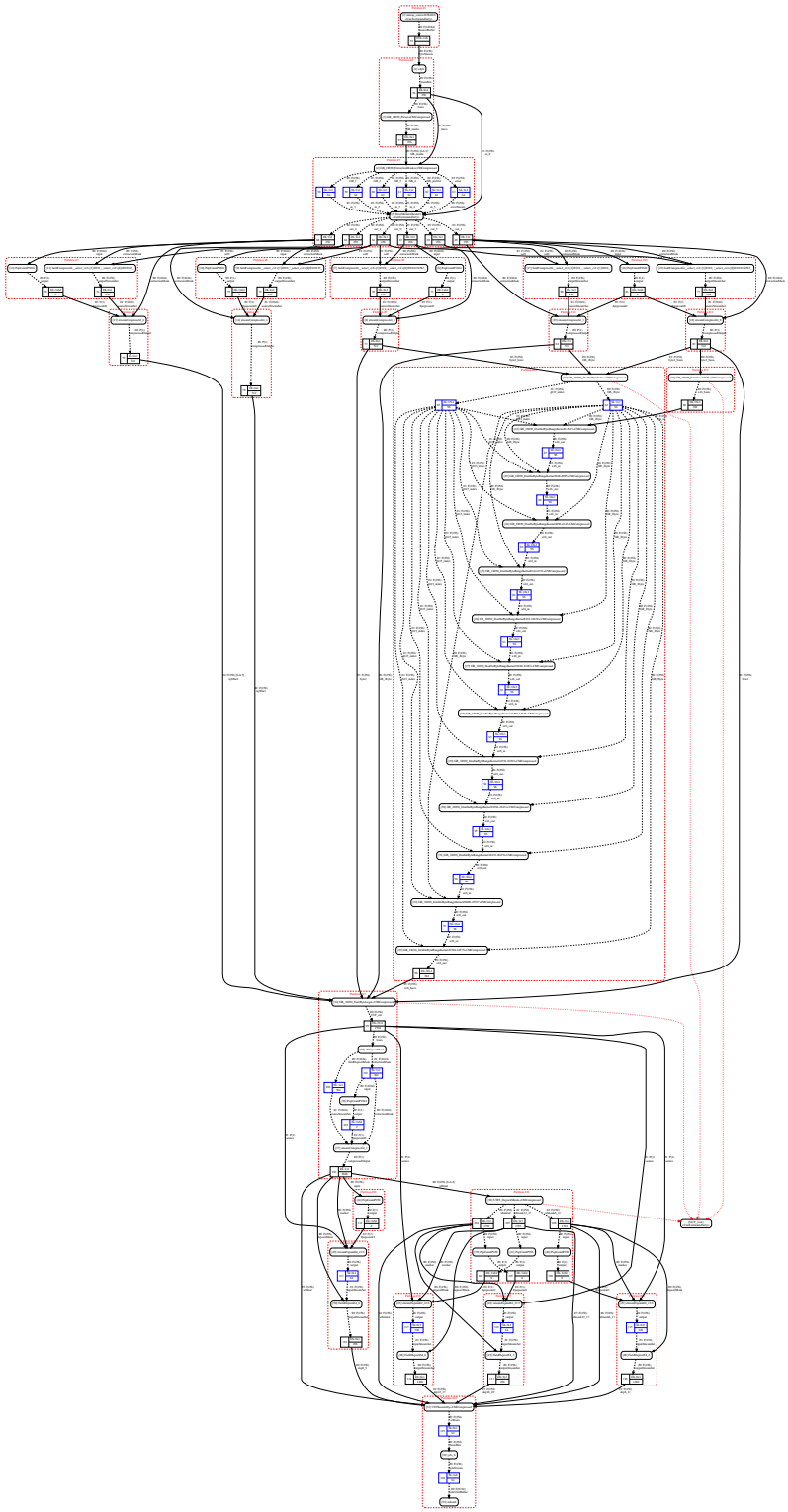


Figure C.5: Pipeline graph for gb18030

C.5 icgrep -colours=never

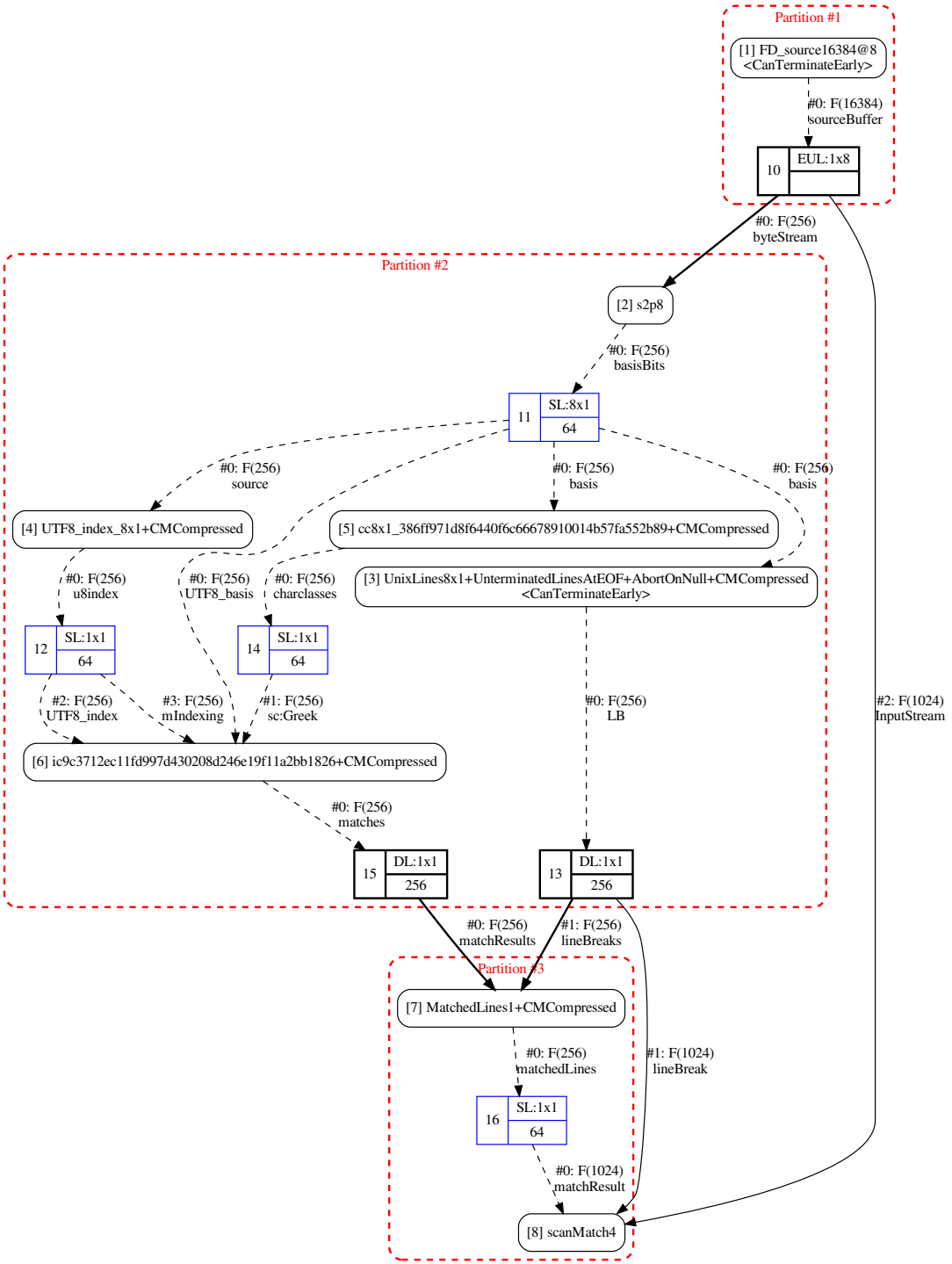


Figure C.6: Sample pipeline graph for icgrep -colours=never

C.6 icgrep -colours=always

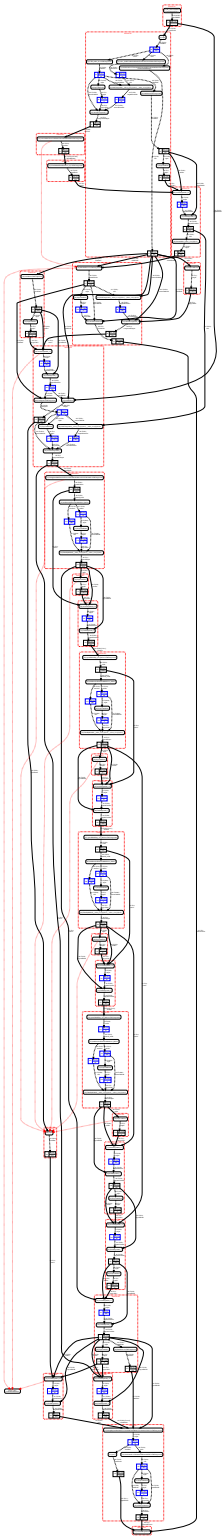


Figure C.7: Sample pipeline graph for icgrep -colours=always

C.7 u8u16

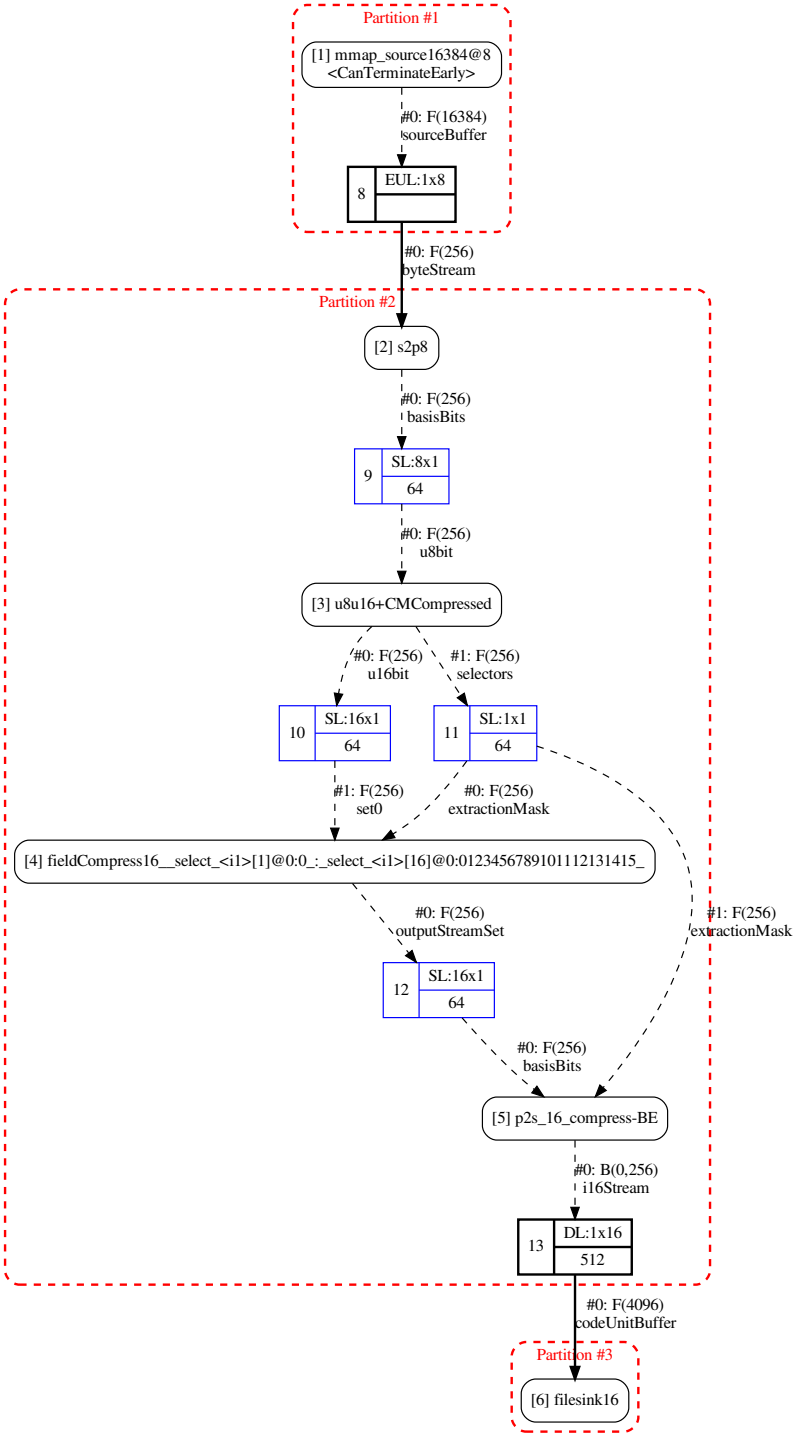


Figure C.8: Pipeline graph for u8u16

C.8 u32u8

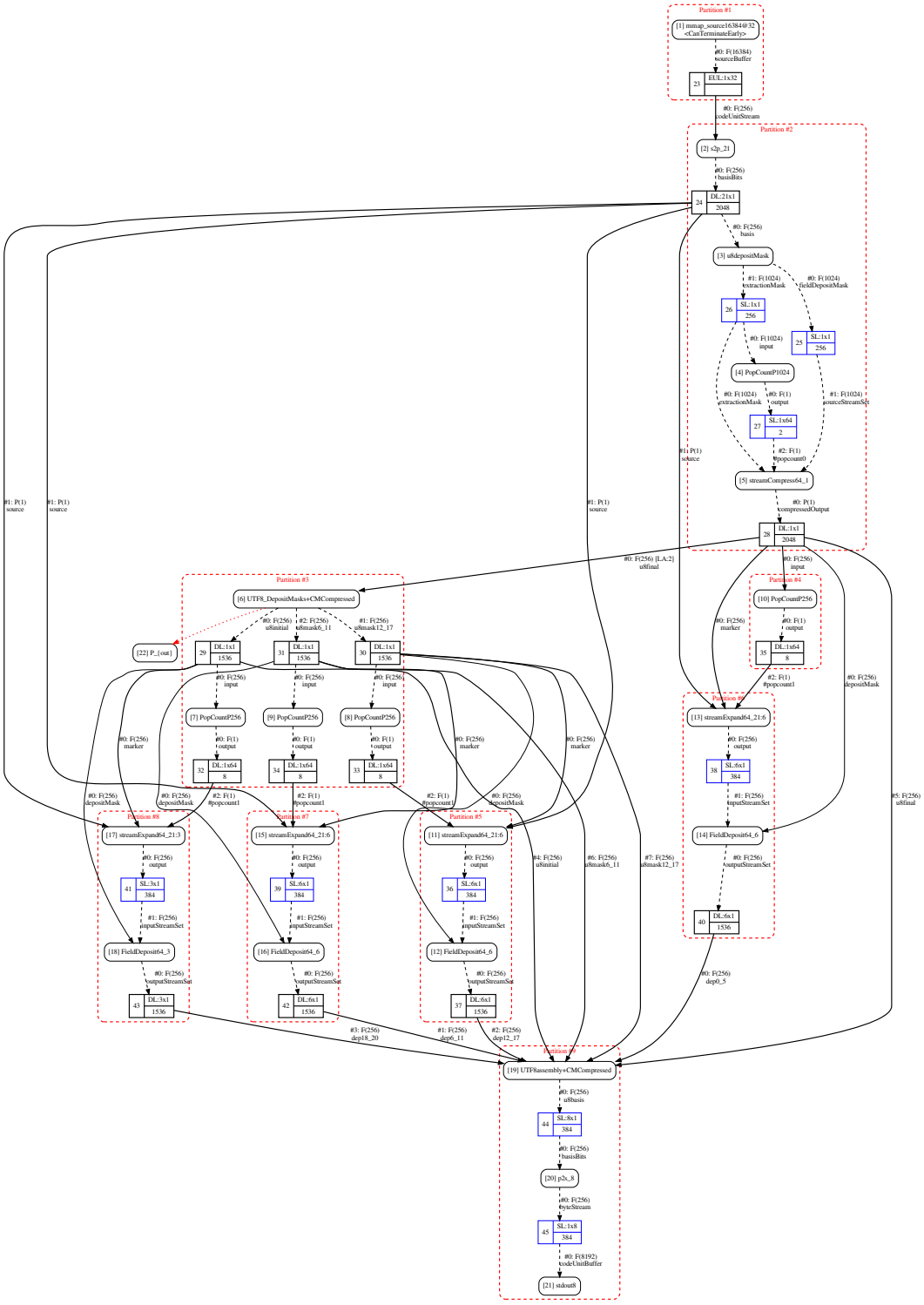


Figure C.9: Pipeline graph for u32u8

C.9 ztf-hash compression

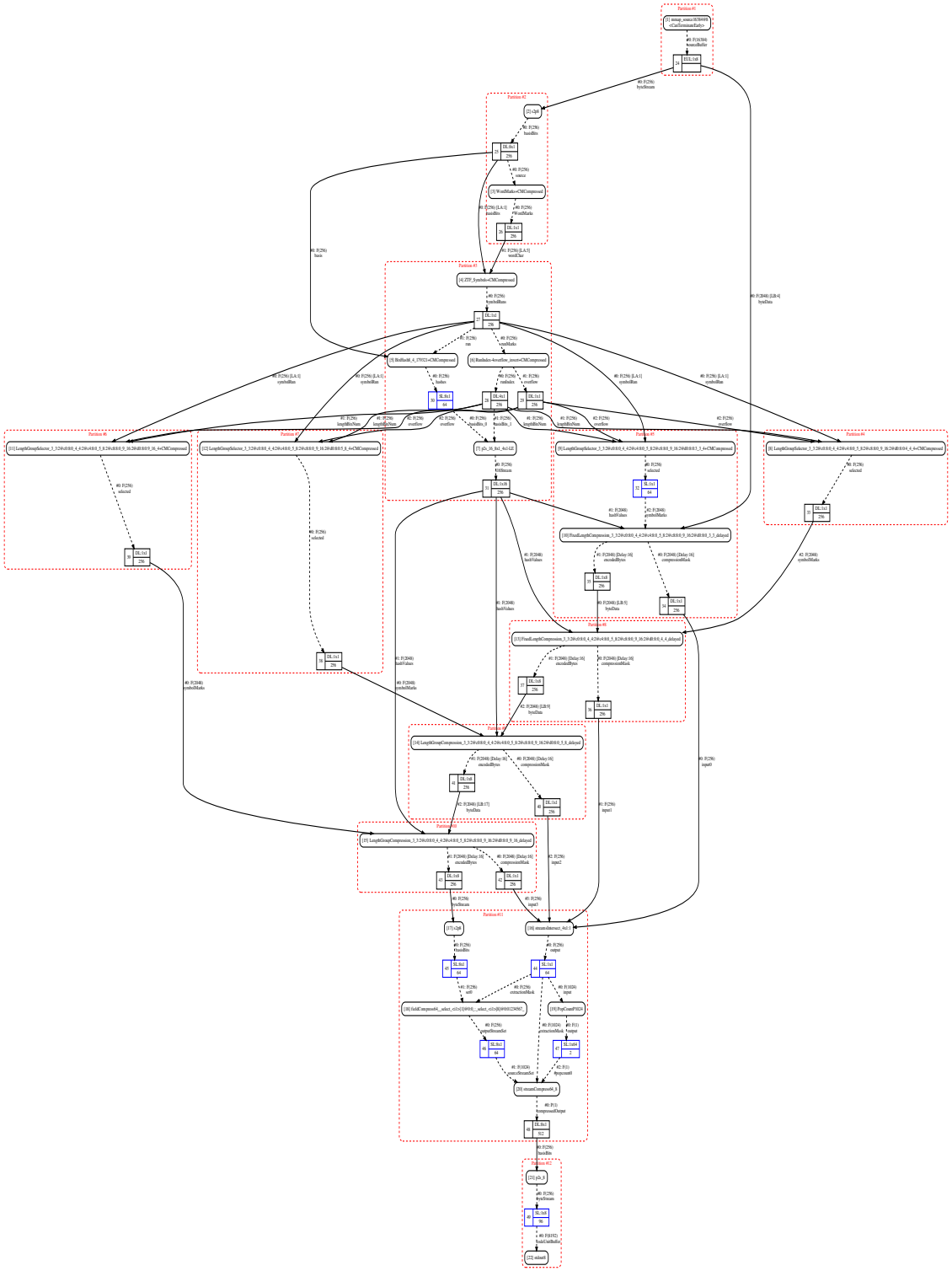


Figure C.10: Pipeline graph for ztf-hash compression

C.10 ztf-hash decompression

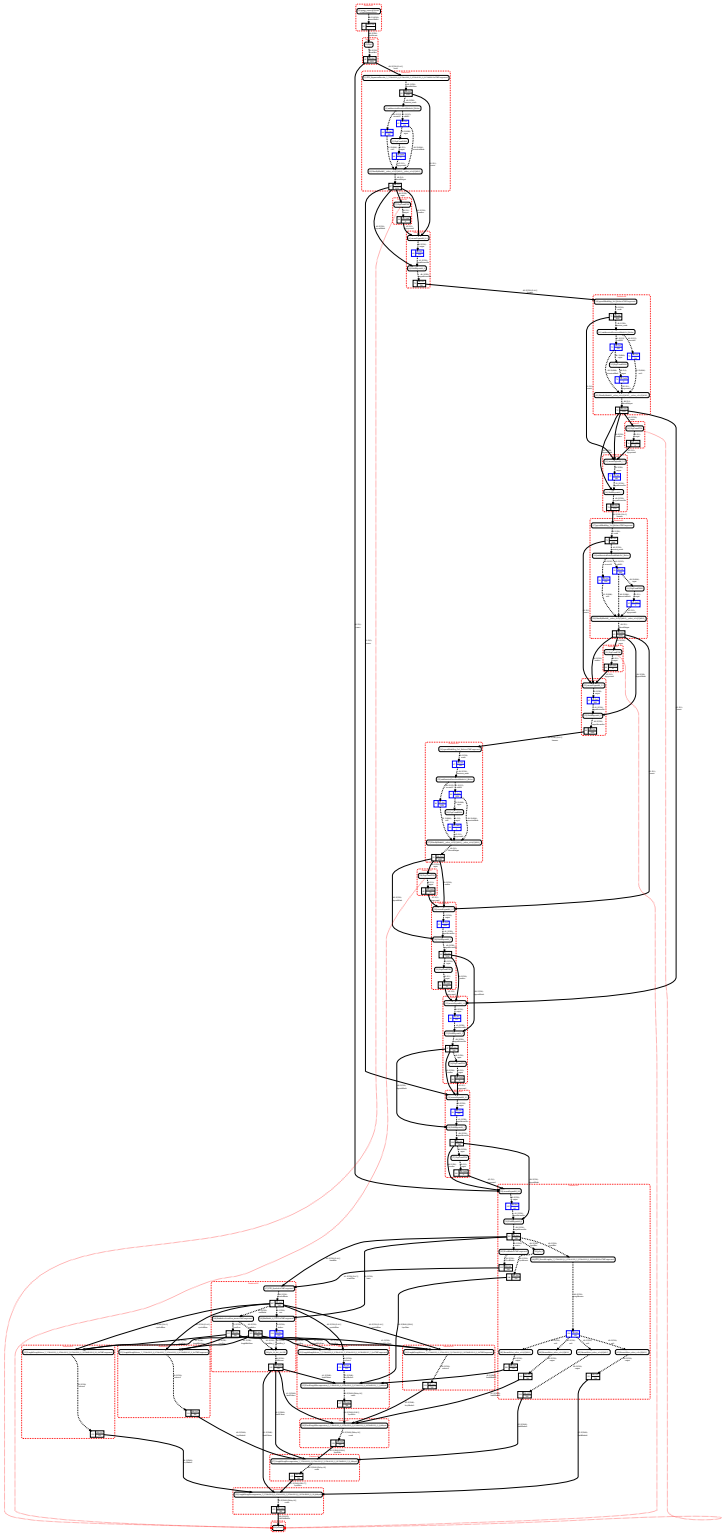


Figure C.11: Pipeline graph for ztf-hash decompression

Appendix D

Detailed evaluation data tables

D.1 Multi-thread speedup evaluation

D.1.1 Case study: `base64`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.92	0.98	0.98	0.99	1.00
4	2	1.25	1.41	1.59	1.70	1.85
4	3	1.69	1.84	1.96	2.07	2.15
4	4	1.46	1.89	2.56	2.97	3.68
16	1	0.84	0.98	0.99	0.99	1.00
16	2	1.37	1.65	1.80	1.88	1.94
16	3	1.90	2.36	2.71	2.78	2.91
16	4	1.37	2.49	2.94	3.19	3.83
32	1	0.84	0.99	0.99	0.99	1.00
32	2	1.51	1.78	1.87	1.93	1.99
32	3	2.26	2.46	2.77	2.91	3.04
32	4	2.15	3.02	3.39	3.77	3.91

Table D.1: `base64` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	34.0	36.3	37.0	37.5	38.8	39.9	40.7	47.3	51.3
Enwiki	34.1	36.3	37.4	39.6	41.2	42.0	40.3	46.9	50.9
Ruwiki	34.1	36.2	37.3	37.6	38.8	39.9	40.2	46.7	50.8
Zhwiki	34.2	36.5	37.5	37.4	38.8	40.1	41.1	47.5	51.6

Table D.2: Relative cost (%) of most expensive kernel in single-threaded execution of `base64`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	32.2	33.9	35.3	35.4	40.4	41.7	38.7	50.0	58.1
Enwiki	32.3	33.9	35.8	38.2	43.7	44.4	38.1	48.6	55.7
Ruwiki	32.4	34.3	36.1	35.3	40.2	41.7	38.1	48.9	55.9
Zhwiki	32.4	34.4	36.0	35.8	40.1	41.6	39.2	50.1	56.3

Table D.3: Relative cost (%) of most expensive kernel in 2-threaded execution of `base64`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	30.4	33.9	34.0	39.1	40.0	43.4	41.4	55.4	54.6
Enwiki	30.7	34.0	34.5	41.4	41.2	43.4	40.8	54.5	54.1
Ruwiki	30.6	34.2	34.3	39.0	39.8	43.5	40.0	54.5	54.4
Zhwiki	30.5	34.2	33.8	38.9	39.9	43.5	40.9	57.1	54.7

Table D.4: Relative cost (%) of most expensive kernel in 3-threaded execution of `base64`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	30.5	34.4	34.9	35.0	43.1	41.6	38.4	53.3	60.2
Enwiki	30.3	34.0	35.2	35.7	45.3	42.4	39.1	52.0	59.8
Ruwiki	30.5	34.1	35.2	35.8	44.6	42.2	39.1	52.7	59.7
Zhwiki	30.3	33.9	34.9	34.7	43.1	41.8	38.6	53.4	60.1

Table D.5: Relative cost (%) of most expensive kernel in 4-threaded execution of `base64`

D.1.2 Case study: `csv2json`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.74	0.95	0.96	0.98	1.00
4	2	1.03	1.27	1.56	1.66	1.88
4	3	1.18	1.67	2.10	2.54	2.78
4	4	1.12	1.81	2.63	3.25	3.70
16	1	0.78	0.99	0.99	0.99	1.00
16	2	1.57	1.80	1.88	1.95	1.98
16	3	2.21	2.63	2.84	2.89	2.96
16	4	2.12	3.24	3.48	3.82	3.95
32	1	0.77	0.95	0.99	1.00	1.00
32	2	1.61	1.88	1.94	1.98	2.00
32	3	2.11	2.72	2.85	2.95	2.98
32	4	2.02	3.14	3.61	3.88	3.97

Table D.6: `csv2json` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Census	13.3	13.7	13.8	15.0	15.9	16.1	14.8	15.6	15.1
Fin	9.9	10.3	10.3	20.9	22.5	22.7	22.3	22.7	22.6
Trade	10.5	10.9	11.0	17.1	18.2	18.4	18.0	18.8	18.5

Table D.7: Relative cost (%) of most expensive kernel in single-threaded execution of `csv2json`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Census	13.2	13.7	13.8	17.9	16.4	16.2	22.7	17.3	16.3
Fin	11.9	10.8	10.4	23.9	25.3	23.4	28.0	27.6	25.5
Trade	10.3	10.9	11.0	20.8	18.9	18.8	16.2	21.6	19.6

Table D.8: Relative cost (%) of most expensive kernel in 2-threaded execution of `csv2json`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Census	12.9	13.7	13.8	18.0	17.4	16.5	26.0	19.3	18.8
Fin	12.6	11.1	11.1	26.4	24.8	23.5	25.4	28.4	24.5
Trade	10.4	10.8	11.0	23.4	20.1	19.2	18.2	22.4	20.4

Table D.9: Relative cost (%) of most expensive kernel in 3-threaded execution of `csv2json`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Census	13.0	13.3	13.5	20.1	17.6	16.7	32.0	22.3	19.3
Fin	14.7	11.3	10.6	29.6	31.8	27.8	26.1	32.6	23.7
Trade	10.2	10.8	10.9	24.8	21.8	20.2	20.0	24.7	21.1

Table D.10: Relative cost (%) of most expensive kernel in 4-threaded execution of `csv2json`

D.1.3 Case study: `editd`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.64	0.90	0.93	0.95	0.99
4	2	1.22	1.60	1.79	1.81	1.90
4	3	1.87	2.37	2.59	2.68	2.84
4	4	2.11	3.01	3.43	3.55	3.77
16	1	0.67	0.98	0.99	0.99	1.00
16	2	0.86	1.75	1.95	1.96	2.01
16	3	2.11	2.60	2.92	2.94	2.96
16	4	0.99	2.96	3.86	3.89	3.93
32	1	0.71	0.99	0.99	1.00	1.00
32	2	1.38	1.85	1.98	1.98	2.00
32	3	0.92	2.36	2.93	2.97	2.99
32	4	0.98	2.85	3.83	3.94	3.99

Table D.11: `editd` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
Dist	4	16	32	4	16	32	4	16	32
1	8.4	8.7	8.8	5.9	5.9	6.1	4.7	4.6	5.5
2	6.8	7.0	6.9	5.5	6.0	5.5	3.9	4.8	7.4
4	4.3	4.3	4.5	4.0	4.2	3.9	2.8	5.1	8.1
8	27.7	29.7	29.2	16.8	16.8	16.8	39.2	40.4	40.1

Table D.12: Relative cost (%) of most expensive kernel in single-threaded execution of `editd`

CONFIG	SSE-4.2			AVX2			AVX-512		
Dist	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
1	8.4	8.8	8.9	6.1	5.9	6.1	4.4	4.5	5.2
2	6.8	6.9	7.0	5.6	6.1	5.6	3.5	4.6	6.9
4	4.2	4.4	5.0	4.0	4.1	4.2	2.6	4.8	7.8
8	36.3	35.0	35.7	23.3	23.9	23.6	49.9	51.4	50.5

Table D.13: Relative cost (%) of most expensive kernel in 2-threaded execution of `editd`

CONFIG	SSE-4.2			AVX2			AVX-512		
Dist	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
1	8.4	8.8	8.9	6.1	6.0	6.2	4.3	4.4	5.1
2	6.9	7.0	7.1	5.7	6.2	5.7	3.6	4.6	7.0
4	4.0	4.4	7.2	4.1	4.2	5.0	2.6	4.8	8.0
8	42.9	41.1	40.8	29.3	29.8	28.9	57.3	59.0	57.2

Table D.14: Relative cost (%) of most expensive kernel in 3-threaded execution of `editd`

CONFIG	SSE-4.2			AVX2			AVX-512		
Dist	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
1	8.3	8.7	8.9	6.0	5.9	6.2	4.3	4.5	5.3
2	6.8	7.0	7.9	5.7	6.3	6.4	3.6	4.3	7.0
4	4.0	5.0	10.3	4.2	4.5	7.2	3.4	4.9	8.7
8	42.8	46.0	41.2	33.0	33.4	33.0	62.2	64.6	61.6

Table D.15: Relative cost (%) of most expensive kernel in 4-threaded execution of `editd`

D.1.4 Case study: gb18030

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.73	0.93	0.98	0.99	1.00
4	2	1.22	1.36	1.74	1.79	1.88
4	3	1.60	2.00	2.58	2.74	2.84
4	4	2.11	2.32	3.36	3.40	3.68
16	1	0.79	0.93	0.97	0.98	1.00
16	2	1.48	1.66	1.90	1.93	1.97
16	3	2.23	2.41	2.83	2.87	2.94
16	4	2.38	3.07	3.68	3.74	3.91
32	1	0.77	0.91	0.98	0.99	1.00
32	2	1.48	1.71	1.90	1.95	1.99
32	3	2.18	2.46	2.84	2.88	2.96
32	4	2.36	3.01	3.69	3.77	3.91

Table D.16: gb18030 speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	6.5	6.5	6.4	9.9	10.1	10.0	9.2	9.3	8.9
Enwiki	6.8	6.9	6.9	7.9	7.5	7.5	6.3	6.9	7.3
Ruwiki	6.3	6.4	6.5	8.1	8.1	8.1	8.8	8.7	9.0
Zhwiki	14.1	14.4	14.5	15.8	16.2	16.3	14.9	15.3	15.5

Table D.17: Relative cost (%) of most expensive kernel in single-threaded execution of gb18030

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	6.7	6.5	6.5	10.1	10.4	10.2	9.1	8.7	9.0
Enwiki	6.6	6.9	6.8	8.0	7.5	7.5	6.2	6.8	7.9
Ruwiki	6.2	6.4	6.4	8.0	8.0	8.1	8.2	8.3	9.1
Zhwiki	14.2	14.6	14.6	15.8	16.3	16.2	14.0	15.3	15.5

Table D.18: Relative cost (%) of most expensive kernel in 2-threaded execution of gb18030

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	7.2	6.5	6.5	9.8	10.5	10.4	8.6	8.7	9.5
Enwiki	6.8	6.8	6.7	7.6	7.5	7.4	6.7	7.2	9.1
Ruwiki	6.2	6.4	6.3	8.1	8.0	8.6	8.5	8.5	9.9
Zhwiki	14.5	14.8	14.7	15.8	16.3	16.3	13.8	15.2	15.4

Table D.19: Relative cost (%) of most expensive kernel in 3-threaded execution of `gb18030`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	7.1	7.5	6.7	10.1	10.4	10.3	8.9	8.7	9.7
Enwiki	6.6	6.7	6.6	8.2	8.0	7.7	6.4	7.2	11.3
Ruwiki	6.4	6.4	6.3	8.2	8.4	8.7	8.0	8.5	10.3
Zhwiki	14.6	14.9	14.7	16.0	16.4	16.3	13.7	15.1	15.0

Table D.20: Relative cost (%) of most expensive kernel in 4-threaded execution of `gb18030`

D.1.5 Case study: `icgrep -colours=never`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.77	0.92	0.94	0.96	0.98
4	2	0.69	1.14	1.31	1.47	1.81
4	3	0.74	1.18	1.34	1.61	2.25
4	4	0.70	1.13	1.34	1.67	3.12
16	1	0.77	0.98	0.99	0.99	1.00
16	2	0.88	1.30	1.51	1.71	1.97
16	3	0.90	1.33	1.61	1.97	2.77
16	4	0.78	1.23	1.54	2.05	3.64
32	1	0.79	0.98	0.99	0.99	1.00
32	2	0.90	1.34	1.54	1.77	1.96
32	3	0.91	1.32	1.64	2.16	2.88
32	4	0.86	1.24	1.54	2.08	3.71

Table D.21: `icgrep` speed up compared to minimum single-thread cost

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	38.4	39.0	39.2	47.8	50.5	51.2	56.9	64.8	66.3
@	Enwiki	38.9	38.9	38.6	48.2	51.3	51.8	56.7	64.5	65.5
@	Ruwiki	38.2	38.4	39.1	47.7	50.4	50.9	57.1	64.6	65.6
@	Zhwiki	37.0	37.0	37.2	47.2	50.2	50.8	55.6	62.9	64.0
Date	Arwiki	47.3	48.9	49.6	49.8	53.4	54.0	50.6	55.1	55.9
Date	Enwiki	49.8	51.9	52.4	54.9	60.4	60.7	58.7	66.0	67.2
Date	Ruwiki	47.5	49.1	49.8	50.2	54.0	54.5	51.4	56.2	56.9
Date	Zhwiki	46.3	48.0	48.6	49.5	53.0	53.5	49.1	53.2	54.1
Email	Arwiki	44.8	46.9	47.1	41.3	43.4	44.0	40.0	43.0	43.4
Email	Enwiki	33.1	34.3	34.3	29.1	30.0	30.2	34.2	36.5	36.8
Email	Ruwiki	47.8	49.9	50.1	45.0	47.9	48.3	44.7	48.1	48.5
Email	Zhwiki	55.3	57.4	57.7	51.8	54.7	55.1	50.4	54.1	54.6
URIOrEmail	Arwiki	30.4	31.3	31.2	37.0	38.6	38.5	50.4	53.2	53.3
URIOrEmail	Enwiki	38.1	40.0	39.9	35.6	37.5	37.4	42.8	48.5	48.7
URIOrEmail	Ruwiki	28.1	29.0	28.8	33.8	35.2	35.2	46.4	49.1	49.1
URIOrEmail	Zhwiki	31.1	31.9	31.9	37.4	38.9	38.9	51.7	54.5	54.6
Xquote	Arwiki	35.1	36.1	36.1	32.4	33.7	33.6	46.0	48.6	48.7
Xquote	Enwiki	46.6	48.6	48.5	42.4	44.8	44.9	33.7	37.4	37.5
Xquote	Ruwiki	35.6	36.6	36.8	31.6	32.3	32.3	39.9	42.2	42.2
Xquote	Zhwiki	32.5	33.2	33.2	30.9	32.2	32.1	44.8	47.3	47.4
\X	Arwiki	77.1	78.0	78.1	56.8	57.8	58.0	47.8	47.8	48.0
\X	Enwiki	78.7	79.6	80.3	57.7	59.2	59.2	47.9	48.5	48.6
\X	Ruwiki	79.0	80.2	80.3	59.3	60.7	61.0	50.7	51.5	51.7
\X	Zhwiki	71.9	73.3	73.7	50.9	52.1	52.4	44.0	44.6	44.8
\p{Greek}	Arwiki	35.8	38.3	38.4	40.0	44.8	45.0	45.5	52.5	53.1
\p{Greek}	Enwiki	39.7	43.4	43.3	41.0	46.8	46.9	44.8	52.1	52.6
\p{Greek}	Ruwiki	33.6	35.5	35.8	35.8	39.9	40.1	40.5	46.2	46.6
\p{Greek}	Zhwiki	28.3	29.7	29.8	32.1	35.1	35.3	37.4	42.3	42.6

Table D.22: Relative cost (%) of most expensive kernel in single-threaded execution of `icgrep -colors=never`

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	40.8	43.8	44.2	47.8	54.6	55.7	60.3	69.4	72.9
@	Enwiki	40.3	42.7	43.4	48.4	55.6	56.8	59.3	68.8	72.9
@	Ruwiki	41.0	44.0	44.6	47.4	54.4	55.6	59.3	69.0	73.1
@	Zhwiki	40.2	40.5	42.3	47.3	53.9	55.0	57.7	67.4	71.3
Date	Arwiki	47.7	51.3	52.0	50.0	56.1	57.2	50.9	56.4	57.8
Date	Enwiki	51.6	55.3	56.9	57.2	65.9	67.4	60.6	70.7	73.3
Date	Ruwiki	47.9	51.7	52.8	50.6	56.9	57.7	52.0	57.8	59.4
Date	Zhwiki	46.6	49.9	50.4	49.7	55.5	56.2	49.6	54.0	55.3
Email	Arwiki	45.1	50.8	51.0	40.8	48.1	48.8	33.8	43.2	48.5
Email	Enwiki	31.6	34.6	35.7	28.9	30.1	30.7	33.9	37.9	38.2
Email	Ruwiki	49.1	54.4	54.0	46.1	52.3	53.0	39.3	50.7	53.4
Email	Zhwiki	61.9	64.9	65.6	55.8	60.5	61.4	46.0	58.7	58.8
URIOrEmail	Arwiki	38.6	39.2	38.9	46.2	48.8	48.6	58.5	63.2	63.5
URIOrEmail	Enwiki	37.5	43.3	43.3	34.1	40.8	41.3	36.0	46.1	45.8
URIOrEmail	Ruwiki	35.4	36.0	35.8	42.4	44.2	44.2	54.6	58.7	58.8
URIOrEmail	Zhwiki	39.3	39.9	39.7	46.7	48.9	49.0	60.2	64.5	64.7
Xquote	Arwiki	32.7	34.1	34.1	41.0	43.1	42.9	54.0	58.3	58.9
Xquote	Enwiki	47.1	50.8	50.6	41.7	46.6	47.0	28.4	33.6	36.7
Xquote	Ruwiki	33.1	35.0	35.1	34.9	37.1	37.4	47.6	51.6	51.8
Xquote	Zhwiki	31.7	32.6	32.3	39.4	41.7	41.9	53.1	57.6	58.0
\X	Arwiki	82.9	84.0	84.0	59.1	61.6	62.0	44.8	46.8	47.5
\X	Enwiki	84.7	85.9	86.1	60.9	63.5	64.2	45.4	47.4	48.2
\X	Ruwiki	84.9	86.1	86.2	62.5	65.5	65.9	48.8	52.1	52.9
\X	Zhwiki	78.7	79.8	80.5	50.9	53.6	53.9	40.2	42.0	42.4
\p{Greek}	Arwiki	35.9	37.5	37.4	38.6	44.5	45.4	37.7	49.2	52.0
\p{Greek}	Enwiki	39.7	43.7	43.3	39.2	46.8	45.6	36.8	49.9	51.3
\p{Greek}	Ruwiki	32.8	33.7	34.4	33.8	37.6	39.3	33.3	42.8	41.8
\p{Greek}	Zhwiki	27.6	26.6	27.0	29.8	31.8	32.4	30.3	36.9	39.1

Table D.23: Relative cost (%) of most expensive kernel in 2-threaded execution of `icgrep -colors=never`

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	46.5	49.2	51.2	58.7	65.3	67.9	74.5	80.7	83.2
@	Enwiki	45.5	48.7	49.7	59.2	66.1	68.2	74.2	80.6	83.1
@	Ruwiki	47.5	50.2	54.1	58.3	64.9	67.4	74.3	80.5	83.1
@	Zhwiki	46.1	46.1	47.1	58.1	64.5	66.9	73.1	79.1	81.5
Date	Arwiki	60.0	63.5	64.6	63.0	66.7	67.1	62.8	65.1	66.4
Date	Enwiki	66.6	70.0	71.2	73.0	78.0	78.5	75.8	82.0	83.4
Date	Ruwiki	60.6	64.2	65.2	63.6	68.0	68.3	63.8	66.6	68.2
Date	Zhwiki	58.6	62.1	62.8	62.2	66.1	66.9	60.4	63.2	64.3
Email	Arwiki	65.8	69.1	69.7	66.0	69.9	71.8	50.9	58.6	62.1
Email	Enwiki	32.6	36.3	36.8	28.6	31.5	31.3	37.4	40.0	41.1
Email	Ruwiki	69.8	73.2	73.3	71.1	75.6	75.9	62.2	69.4	71.1
Email	Zhwiki	76.5	79.7	79.8	74.3	77.6	77.7	67.7	75.1	76.4
URIOrEmail	Arwiki	42.3	44.1	45.1	50.9	54.0	55.6	64.2	68.7	71.0
URIOrEmail	Enwiki	43.4	49.5	49.3	42.1	46.5	46.0	37.5	45.4	45.3
URIOrEmail	Ruwiki	39.3	40.4	41.5	47.1	49.5	50.5	60.0	64.2	66.7
URIOrEmail	Zhwiki	43.6	45.1	45.9	51.8	54.9	55.8	65.7	70.1	72.0
Xquote	Arwiki	36.7	39.0	39.3	43.7	45.8	45.8	60.0	63.9	66.3
Xquote	Enwiki	59.8	63.7	63.3	54.6	59.2	59.3	29.7	38.2	35.0
Xquote	Ruwiki	37.7	40.1	40.5	37.7	40.2	40.4	53.5	57.5	59.8
Xquote	Zhwiki	34.2	35.7	35.8	43.0	46.0	46.6	59.4	63.6	65.5
\X	Arwiki	86.9	87.4	87.7	67.6	69.1	69.2	54.3	55.6	56.1
\X	Enwiki	88.7	89.3	89.6	70.2	71.7	72.0	56.5	58.1	58.5
\X	Ruwiki	88.6	89.3	89.4	70.9	72.5	72.8	58.4	60.4	61.1
\X	Zhwiki	84.1	84.8	85.2	61.8	63.5	63.8	50.0	51.7	52.3
\p{Greek}	Arwiki	35.3	37.2	37.8	37.1	42.3	42.9	39.5	49.3	48.3
\p{Greek}	Enwiki	40.4	42.8	43.3	38.7	46.3	45.7	38.9	50.4	49.1
\p{Greek}	Ruwiki	32.3	33.5	35.3	32.5	35.8	37.3	34.6	44.4	40.5
\p{Greek}	Zhwiki	26.9	26.5	29.1	29.0	30.1	32.7	32.0	37.4	37.4

Table D.24: Relative cost (%) of most expensive kernel in 3-threaded execution of `icgrep -colors=never`

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	42.2	53.9	60.0	66.4	73.4	74.4	80.6	85.1	87.1
@	Enwiki	41.6	51.8	59.9	66.0	74.2	75.9	80.8	85.0	87.6
@	Ruwiki	42.2	50.3	60.1	66.7	73.3	74.5	80.6	84.9	87.0
@	Zhwiki	40.7	52.2	57.1	63.5	72.9	73.6	79.4	83.8	86.1
Date	Arwiki	66.2	69.1	70.3	68.1	71.0	71.6	66.9	68.8	70.0
Date	Enwiki	73.6	76.5	77.8	79.2	82.9	83.5	81.8	86.2	87.5
Date	Ruwiki	66.8	70.3	71.5	69.0	72.7	73.4	67.8	70.6	72.3
Date	Zhwiki	65.0	68.3	69.6	67.3	70.2	71.2	64.1	66.6	68.3
Email	Arwiki	76.6	79.6	79.7	77.9	79.6	79.9	66.5	74.9	75.6
Email	Enwiki	40.9	45.5	48.3	34.4	38.8	38.1	38.7	41.6	42.7
Email	Ruwiki	79.8	82.5	82.6	80.5	82.9	82.7	75.3	80.6	81.3
Email	Zhwiki	84.3	86.0	86.0	82.2	83.8	84.0	79.3	83.3	83.8
URIOrEmail	Arwiki	43.6	41.0	44.8	53.9	55.7	55.7	68.4	73.9	75.4
URIOrEmail	Enwiki	55.9	62.7	62.5	53.4	58.0	59.8	32.5	37.7	46.3
URIOrEmail	Ruwiki	40.5	41.8	41.6	50.2	51.9	51.6	62.5	69.1	71.0
URIOrEmail	Zhwiki	45.6	47.0	47.0	55.4	57.8	57.6	68.7	74.9	76.3
Xquote	Arwiki	43.7	47.5	47.7	43.8	44.1	44.3	64.0	68.9	70.4
Xquote	Enwiki	69.7	73.3	73.0	65.2	69.0	69.2	39.5	41.8	45.1
Xquote	Ruwiki	44.8	49.3	49.5	38.4	40.6	40.7	57.2	62.0	63.7
Xquote	Zhwiki	38.2	42.5	42.9	44.1	45.4	45.5	63.2	68.4	69.7
\X	Arwiki	89.1	89.6	89.6	71.6	72.8	73.0	59.8	60.3	62.8
\X	Enwiki	90.7	91.0	91.1	74.9	75.9	76.0	62.9	63.6	63.9
\X	Ruwiki	90.4	90.9	91.1	74.8	76.1	76.3	63.7	64.9	65.3
\X	Zhwiki	86.9	87.3	87.4	67.1	68.5	68.7	56.3	57.2	57.5
\p{Greek}	Arwiki	31.8	32.0	36.5	32.7	36.4	41.3	35.5	39.1	49.5
\p{Greek}	Enwiki	39.7	40.6	43.4	36.6	41.3	44.5	36.4	46.1	50.3
\p{Greek}	Ruwiki	31.0	32.5	30.7	28.8	31.9	35.8	29.7	33.7	39.4
\p{Greek}	Zhwiki	38.4	33.8	27.8	26.0	28.6	30.5	27.4	31.5	35.6

Table D.25: Relative cost (%) of most expensive kernel in 4-threaded execution of `icgrep -colors=never`

D.1.6 Case study: `icgrep -colours=always`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.73	0.90	0.92	0.93	0.96
4	2	0.71	0.97	1.35	1.56	1.80
4	3	0.90	1.19	1.74	2.13	2.66
4	4	0.97	1.44	2.05	2.61	3.51
16	1	0.79	0.98	0.99	0.99	1.00
16	2	1.21	1.47	1.66	1.80	1.96
16	3	1.47	2.06	2.33	2.67	2.92
16	4	1.62	2.28	2.69	3.21	3.87
32	1	0.79	0.99	0.99	1.00	1.00
32	2	1.29	1.57	1.75	1.87	1.98
32	3	1.56	2.14	2.39	2.72	2.96
32	4	1.68	2.45	2.85	3.40	3.94

Table D.26: `icgrep` speed up compared to minimum single-thread cost

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	13.3	14.2	14.4	14.8	16.5	17.0	21.6	26.7	28.0
@	Enwiki	13.3	14.3	14.5	14.9	16.8	17.4	21.0	26.6	28.0
@	Ruwiki	13.3	14.1	14.3	14.6	16.3	16.9	21.6	26.5	27.9
@	Zhwiki	13.2	14.1	14.2	14.5	16.2	16.7	20.8	25.7	27.0
Date	Arwiki	10.6	11.1	11.3	16.2	18.2	18.7	18.9	22.3	23.2
Date	Enwiki	12.3	13.2	13.4	18.0	20.9	21.5	20.9	25.9	27.2
Date	Ruwiki	10.4	11.0	11.1	16.1	18.0	18.4	18.9	22.3	23.2
Date	Zhwiki	9.9	10.4	10.5	15.8	17.5	17.9	17.9	21.0	21.9
Email	Arwiki	8.9	9.4	9.5	12.7	14.7	15.0	16.9	21.2	21.9
Email	Enwiki	9.0	9.5	9.6	12.6	14.8	15.2	16.2	20.8	21.5
Email	Ruwiki	8.8	9.4	9.5	12.5	14.5	14.9	16.8	20.8	21.6
Email	Zhwiki	8.7	9.3	9.3	12.4	14.4	14.7	16.1	20.2	20.8
URIOrEmail	Arwiki	5.4	5.7	5.8	10.7	12.0	12.3	18.7	21.7	22.4
URIOrEmail	Enwiki	10.0	10.7	10.7	12.2	13.6	13.8	15.4	19.4	20.1
URIOrEmail	Ruwiki	5.4	5.7	5.8	9.6	10.6	10.9	16.6	19.4	19.9
URIOrEmail	Zhwiki	5.2	5.3	5.4	10.9	12.3	12.4	19.1	22.2	22.8
Xquote	Arwiki	6.6	6.8	6.9	10.1	11.3	11.4	17.9	20.8	21.5
Xquote	Enwiki	10.9	11.4	11.5	15.4	17.4	17.7	13.1	16.2	16.9
Xquote	Ruwiki	6.9	7.2	7.3	10.1	11.0	11.3	15.4	18.0	18.7
Xquote	Zhwiki	6.6	6.9	6.9	10.3	11.4	11.6	18.2	21.2	21.8
\X	Arwiki	8.4	8.3	8.2	18.5	20.2	20.4	28.6	31.6	32.1
\X	Enwiki	10.5	10.5	10.5	23.1	25.0	25.3	33.6	37.0	37.4
\X	Ruwiki	6.7	6.4	6.6	14.3	15.8	16.0	23.3	26.2	26.6
\X	Zhwiki	26.7	28.3	28.6	23.2	25.0	25.1	21.5	23.4	23.7
\p{Greek}	Arwiki	8.0	8.4	8.5	13.3	15.5	16.0	17.0	21.5	22.6
\p{Greek}	Enwiki	8.2	8.7	8.8	13.5	15.8	16.4	16.5	21.4	22.5
\p{Greek}	Ruwiki	7.6	8.1	8.2	12.5	14.6	15.1	16.0	19.9	21.0
\p{Greek}	Zhwiki	7.4	7.8	7.9	12.0	13.9	14.3	15.1	18.8	19.8

Table D.27: Relative cost (%) of most expensive kernel in single-threaded execution of `icgrep -colors=always`

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	11.8	13.4	13.8	12.1	14.9	16.7	15.3	24.4	28.6
@	Enwiki	11.9	13.6	14.0	12.1	15.6	17.3	15.1	24.4	28.6
@	Ruwiki	11.7	13.1	13.6	12.0	14.8	16.5	15.4	24.5	28.9
@	Zhwiki	11.6	13.1	13.6	12.0	14.6	16.4	14.5	23.1	27.1
Date	Arwiki	8.5	9.9	10.3	13.0	17.0	18.2	13.1	18.9	21.9
Date	Enwiki	11.1	12.5	12.7	15.3	20.9	21.7	15.6	24.1	27.9
Date	Ruwiki	8.3	9.6	10.0	12.6	16.6	18.1	13.1	18.9	21.9
Date	Zhwiki	8.4	9.1	9.5	12.2	16.2	17.4	12.1	17.7	20.7
Email	Arwiki	8.3	9.2	9.2	10.9	13.5	14.4	11.7	17.9	21.0
Email	Enwiki	8.4	9.2	9.3	11.1	13.4	13.9	11.4	17.5	20.1
Email	Ruwiki	8.2	9.0	9.1	10.9	13.9	13.7	11.7	18.5	19.9
Email	Zhwiki	8.2	8.9	9.1	10.7	13.2	13.6	11.1	17.4	19.2
URIOrEmail	Arwiki	6.5	7.5	7.5	11.0	17.6	18.3	16.1	20.4	23.1
URIOrEmail	Enwiki	9.5	10.4	10.5	10.9	13.5	14.5	11.0	16.4	19.0
URIOrEmail	Ruwiki	5.5	6.5	6.5	9.6	14.8	15.9	15.0	22.4	25.1
URIOrEmail	Zhwiki	6.8	6.2	7.4	10.2	13.8	14.5	14.1	21.1	30.3
Xquote	Arwiki	6.2	6.9	7.1	10.5	12.9	16.9	14.7	17.2	28.7
Xquote	Enwiki	10.5	11.5	12.0	13.6	17.4	18.3	9.6	13.4	15.8
Xquote	Ruwiki	6.6	7.2	7.5	11.1	10.3	14.4	11.7	20.3	22.8
Xquote	Zhwiki	6.3	7.0	7.1	10.4	16.4	17.3	15.0	21.8	27.9
\X	Arwiki	11.3	11.2	11.1	23.5	26.5	26.9	34.0	39.5	40.7
\X	Enwiki	14.4	14.2	14.3	30.4	32.6	33.8	40.6	46.0	46.6
\X	Ruwiki	9.0	8.8	8.8	18.8	21.6	22.2	28.2	33.8	35.0
\X	Zhwiki	27.4	28.6	28.7	22.3	24.3	24.3	26.9	30.7	31.4
\p{Greek}	Arwiki	7.4	8.1	8.3	11.3	14.3	15.6	11.7	18.4	21.9
\p{Greek}	Enwiki	7.6	8.4	8.6	11.4	14.2	15.4	11.4	18.0	21.1
\p{Greek}	Ruwiki	7.0	7.7	7.9	10.5	13.4	14.2	11.1	17.3	19.1
\p{Greek}	Zhwiki	6.9	7.5	7.7	10.1	12.9	13.2	10.5	16.0	17.7

Table D.28: Relative cost (%) of most expensive kernel in 2-threaded execution of `icgrep -colors=always`

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	11.4	13.6	14.0	12.3	14.4	15.0	15.0	26.4	27.4
@	Enwiki	11.6	13.9	14.2	12.6	14.8	15.4	14.2	26.5	27.6
@	Ruwiki	11.3	13.4	13.8	12.2	14.3	14.7	15.1	26.4	27.5
@	Zhwiki	11.1	13.2	13.6	12.0	14.2	15.0	14.0	25.0	26.8
Date	Arwiki	11.1	10.5	9.8	12.8	16.8	18.1	12.5	19.4	21.1
Date	Enwiki	10.9	12.9	13.2	16.4	19.9	20.9	15.0	25.3	25.7
Date	Ruwiki	12.0	10.6	9.5	12.5	17.0	18.4	12.5	19.7	21.6
Date	Zhwiki	14.1	10.3	8.9	12.0	16.7	18.3	11.4	18.4	20.8
Email	Arwiki	8.3	9.2	9.4	11.0	13.6	14.3	11.2	17.9	19.6
Email	Enwiki	8.4	9.3	9.4	11.1	13.8	14.4	10.9	17.6	19.4
Email	Ruwiki	8.2	9.1	9.3	10.9	13.5	14.1	11.2	17.4	19.5
Email	Zhwiki	8.1	9.0	9.2	10.6	13.3	14.0	10.5	16.8	18.7
URIOrEmail	Arwiki	7.6	8.2	9.6	14.0	17.8	23.4	19.4	20.0	25.0
URIOrEmail	Enwiki	9.6	10.5	10.6	11.2	13.4	14.0	10.8	16.5	18.2
URIOrEmail	Ruwiki	8.1	7.9	8.4	12.7	17.8	19.9	19.6	23.2	24.6
URIOrEmail	Zhwiki	7.7	7.3	9.6	13.9	15.4	16.4	16.7	21.7	36.1
Xquote	Arwiki	8.2	7.1	9.0	14.7	14.6	21.6	20.9	17.5	34.5
Xquote	Enwiki	10.6	11.5	11.8	14.0	17.4	17.9	9.7	13.6	15.4
Xquote	Ruwiki	8.1	7.4	7.9	15.9	10.1	18.3	17.3	23.2	22.0
Xquote	Zhwiki	8.2	8.8	9.1	15.9	20.2	22.0	20.6	24.2	30.5
\X	Arwiki	14.2	13.6	13.5	27.8	31.8	32.3	40.4	47.0	47.3
\X	Enwiki	18.6	17.9	17.5	37.1	39.4	40.2	48.2	54.9	54.4
\X	Ruwiki	11.2	11.1	11.1	22.8	26.5	27.3	32.7	40.5	41.9
\X	Zhwiki	30.8	31.4	30.9	22.7	24.9	24.3	30.0	36.8	37.4
\p{Greek}	Arwiki	7.4	8.2	8.3	11.2	14.1	15.0	11.0	18.1	20.4
\p{Greek}	Enwiki	7.6	8.5	8.7	11.4	14.6	15.5	10.8	17.9	20.4
\p{Greek}	Ruwiki	6.9	7.6	7.9	10.3	13.1	13.8	10.4	16.7	18.7
\p{Greek}	Zhwiki	6.7	7.5	7.7	10.0	12.5	13.3	9.9	15.6	17.6

Table D.29: Relative cost (%) of most expensive kernel in 3-threaded execution of `icgrep -colors=always`

CONFIG		SSE-4.2			AVX2			AVX-512		
Expression	**File**	**4**	**16**	**32**	**4**	**16**	**32**	**4**	**16**	**32**
@	Arwiki	11.2	12.4	12.3	11.9	15.0	15.8	13.1	30.5	35.3
@	Enwiki	11.5	12.7	14.0	11.5	16.3	16.7	12.5	30.8	36.0
@	Ruwiki	11.2	12.7	13.2	12.2	16.4	15.1	13.2	29.9	35.6
@	Zhwiki	10.9	12.1	13.2	11.8	14.5	15.8	12.1	28.0	33.8
Date	Arwiki	15.5	9.9	9.2	12.5	17.1	19.3	14.3	21.3	23.5
Date	Enwiki	10.7	12.1	13.0	15.0	21.8	22.9	13.2	30.2	29.9
Date	Ruwiki	16.8	10.5	8.8	12.4	16.7	19.2	14.4	20.5	24.2
Date	Zhwiki	19.5	10.7	8.4	14.9	16.9	19.3	14.1	19.1	23.2
Email	Arwiki	8.2	9.8	9.2	9.8	12.7	13.8	9.9	23.5	21.6
Email	Enwiki	8.2	9.5	9.3	9.9	13.1	13.9	9.6	18.6	20.9
Email	Ruwiki	7.9	9.1	9.0	9.7	12.3	13.8	9.8	18.2	20.6
Email	Zhwiki	8.1	8.7	9.6	9.5	12.1	13.8	9.2	16.8	19.8
URIOrEmail	Arwiki	9.4	9.0	11.5	15.1	17.0	27.8	21.2	18.3	26.6
URIOrEmail	Enwiki	9.6	12.0	10.4	11.5	15.7	14.9	9.5	16.9	19.5
URIOrEmail	Ruwiki	10.9	9.1	8.9	15.5	20.2	20.5	22.0	20.3	25.1
URIOrEmail	Zhwiki	9.7	8.0	11.5	16.8	16.8	18.0	18.6	22.5	40.5
Xquote	Arwiki	10.9	7.6	10.7	19.9	15.8	25.4	24.1	19.8	39.0
Xquote	Enwiki	11.6	13.1	12.1	15.2	19.9	19.5	8.6	13.2	15.6
Xquote	Ruwiki	10.4	7.8	8.9	21.3	13.2	21.3	18.8	21.7	20.3
Xquote	Zhwiki	10.0	10.4	11.0	21.6	23.3	25.7	23.1	26.1	32.7
\X	Arwiki	16.2	16.0	15.8	31.5	36.2	36.4	45.9	52.4	53.0
\X	Enwiki	20.6	20.7	20.7	41.8	43.8	44.4	54.0	59.1	59.4
\X	Ruwiki	13.4	13.3	13.3	26.2	30.9	31.4	36.9	46.3	47.5
\X	Zhwiki	38.0	38.5	38.4	26.1	29.2	28.8	33.1	41.4	42.1
\p{Greek}	Arwiki	7.2	8.0	8.2	9.9	13.0	15.0	9.8	18.3	22.2
\p{Greek}	Enwiki	7.4	8.2	8.5	10.1	13.1	15.3	9.7	18.7	21.6
\p{Greek}	Ruwiki	6.7	7.4	7.7	9.1	12.0	13.5	9.2	16.6	19.2
\p{Greek}	Zhwiki	6.6	7.2	7.5	8.9	12.5	13.1	8.7	15.2	17.8

Table D.30: Relative cost (%) of most expensive kernel in 4-threaded execution of `icgrep -colors=always`

D.1.7 Case study: `u8u16`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.84	0.98	0.98	0.99	1.00
4	2	1.55	1.70	1.79	1.85	1.91
4	3	2.06	2.17	2.52	2.58	2.64
4	4	1.83	2.54	2.78	2.94	3.26
16	1	0.93	0.96	0.98	0.99	1.00
16	2	1.54	1.83	1.87	1.93	1.98
16	3	2.15	2.44	2.67	2.90	2.94
16	4	1.80	2.97	3.34	3.45	3.87
32	1	0.93	0.97	0.99	0.99	1.00
32	2	1.53	1.86	1.89	1.96	1.99
32	3	2.36	2.79	2.89	2.96	2.99
32	4	2.30	3.25	3.48	3.62	3.95

Table D.31: `u8u16v2` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	45.6	48.5	49.0	48.2	50.2	50.3	35.9	42.5	43.7
Enwiki	43.7	47.8	48.4	47.0	49.5	49.8	34.0	41.9	43.1
Ruwiki	44.7	47.4	48.1	47.6	49.5	49.7	36.0	42.0	43.2
Zhwiki	44.2	46.6	47.4	47.3	49.1	49.3	36.2	42.0	43.2

Table D.32: Relative cost (%) of most expensive kernel in single-threaded execution of `u8u16`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	43.1	46.5	47.4	45.5	48.9	49.2	32.8	39.2	41.5
Enwiki	41.6	45.9	46.5	44.5	48.3	48.6	31.0	39.0	40.8
Ruwiki	41.9	45.3	46.4	45.1	48.4	48.9	33.1	39.2	41.2
Zhwiki	41.6	43.8	45.4	44.4	47.9	48.2	33.1	39.4	41.3

Table D.33: Relative cost (%) of most expensive kernel in 2-threaded execution of `u8u16`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	40.7	47.6	49.0	43.5	49.1	50.2	29.9	37.0	42.0
Enwiki	39.3	42.3	47.1	42.1	45.3	47.4	28.8	36.3	36.4
Ruwiki	39.3	46.9	47.9	42.9	48.9	49.7	29.8	36.4	42.6
Zhwiki	37.4	46.2	46.8	42.3	48.4	49.5	30.3	36.8	42.4

Table D.34: Relative cost (%) of most expensive kernel in 3-threaded execution of `u8u16`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	36.2	43.4	47.3	39.1	44.8	45.9	27.5	34.2	38.8
Enwiki	34.1	38.9	45.4	37.8	43.1	44.1	26.6	37.3	38.1
Ruwiki	35.3	41.0	46.2	38.5	45.3	46.3	27.7	34.5	39.0
Zhwiki	43.9	49.0	39.6	37.7	45.0	46.3	27.9	34.9	39.0

Table D.35: Relative cost (%) of most expensive kernel in 4-threaded execution of `u8u16`

D.1.8 Case study: `u32u8`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.86	0.97	0.99	0.99	1.00
4	2	1.27	1.54	1.73	1.84	1.91
4	3	1.83	2.09	2.62	2.78	2.84
4	4	1.87	2.47	3.13	3.53	3.80
16	1	0.86	0.97	0.99	0.99	1.00
16	2	1.72	1.83	1.96	1.97	1.99
16	3	2.30	2.65	2.92	2.93	2.97
16	4	2.64	3.38	3.80	3.87	3.92
32	1	0.94	0.98	0.99	0.99	1.00
32	2	1.76	1.89	1.96	1.98	2.00
32	3	2.17	2.65	2.87	2.90	2.93
32	4	2.25	2.82	3.72	3.78	3.87

Table D.36: `u32u8v2` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	13.2	13.4	13.5	26.0	26.8	27.1	29.8	33.2	33.8
Enwiki	13.7	14.0	14.2	27.9	29.2	29.4	31.9	36.7	37.7
Ruwiki	13.6	13.8	13.9	24.5	25.4	25.5	27.9	31.1	31.6
Zhwiki	13.6	13.9	13.9	24.0	24.9	25.0	27.8	30.4	31.1

Table D.37: Relative cost (%) of most expensive kernel in single-threaded execution of u32u8

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	12.8	13.3	13.3	24.4	26.4	26.4	28.1	32.5	34.3
Enwiki	13.2	13.9	14.0	26.2	28.5	28.8	29.8	35.3	37.5
Ruwiki	13.3	13.8	13.7	23.3	24.9	24.9	26.2	30.5	32.3
Zhwiki	13.3	13.8	13.7	22.5	24.4	24.5	26.0	30.4	33.3

Table D.38: Relative cost (%) of most expensive kernel in 2-threaded execution of u32u8

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	12.8	13.3	13.2	24.0	26.2	27.2	28.4	33.2	37.2
Enwiki	13.5	13.9	13.7	26.5	28.4	29.8	29.1	36.4	42.5
Ruwiki	13.3	13.7	13.6	22.7	24.7	25.6	25.8	31.1	34.9
Zhwiki	13.1	13.7	13.4	22.3	24.3	25.3	25.8	32.5	37.2

Table D.39: Relative cost (%) of most expensive kernel in 3-threaded execution of u32u8

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	12.4	13.2	13.1	23.1	26.3	27.9	28.2	34.8	45.9
Enwiki	13.1	13.7	13.6	24.0	28.4	30.6	29.8	38.9	49.2
Ruwiki	12.4	13.6	13.5	22.4	24.9	26.3	27.4	32.8	43.9
Zhwiki	12.9	13.4	13.3	21.6	24.0	25.7	26.3	34.6	44.7

Table D.40: Relative cost (%) of most expensive kernel in 4-threaded execution of u32u8

D.1.9 Case study: `ztf-hash compression`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.70	0.93	0.94	0.95	0.96
4	2	1.27	1.39	1.72	1.75	1.83
4	3	1.81	1.99	2.51	2.60	2.70
4	4	2.14	2.30	3.17	3.33	3.58
16	1	0.69	0.98	0.98	0.99	1.00
16	2	1.56	1.68	1.91	1.93	1.96
16	3	2.29	2.40	2.83	2.89	2.92
16	4	2.53	2.94	3.62	3.74	3.88
32	1	0.75	0.94	0.99	1.00	1.00
32	2	1.72	1.80	1.94	1.96	1.99
32	3	2.54	2.61	2.89	2.95	2.97
32	4	2.24	3.24	3.72	3.82	3.94

Table D.41: `ztfc` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	16.7	17.5	17.9	21.3	22.1	22.4	26.6	28.2	28.1
Enwiki	19.7	20.8	21.1	24.4	25.5	25.8	29.9	32.2	32.1
Ruwiki	12.9	13.3	13.3	15.5	16.1	16.2	19.7	21.0	21.3
Zhwiki	18.5	18.4	18.5	16.7	17.0	17.2	20.3	21.6	21.9

Table D.42: Relative cost (%) of most expensive kernel in single-threaded execution of `ztf compression`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	16.7	17.5	17.8	20.8	22.1	22.3	23.7	26.7	27.5
Enwiki	19.9	20.6	21.0	24.0	25.4	25.8	26.5	30.0	31.1
Ruwiki	12.4	13.1	13.1	15.2	16.3	16.4	18.5	20.6	21.1
Zhwiki	18.9	18.7	18.5	16.9	17.2	17.2	20.8	21.7	22.1

Table D.43: Relative cost (%) of most expensive kernel in 2-threaded execution of `ztf compression`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	16.9	17.6	17.8	21.0	22.3	22.5	24.3	26.8	28.0
Enwiki	19.8	20.8	21.1	24.3	25.7	25.9	27.2	31.0	32.6
Ruwiki	12.4	13.1	13.2	15.7	16.3	16.5	18.6	20.7	21.1
Zhwiki	19.4	18.7	18.7	17.6	17.6	17.6	21.2	23.2	23.2

Table D.44: Relative cost (%) of most expensive kernel in 3-threaded execution of `ztf compression`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	17.2	18.5	18.2	21.4	23.1	23.2	25.8	29.8	32.3
Enwiki	21.1	22.0	21.7	26.2	29.5	30.1	29.7	40.4	43.0
Ruwiki	12.2	13.1	13.2	16.8	16.7	16.7	20.2	22.6	21.8
Zhwiki	21.7	20.2	19.5	18.8	18.7	18.5	23.8	26.4	25.9

Table D.45: Relative cost (%) of most expensive kernel in 4-threaded execution of `ztf compression`

D.1.10 Case study: `ztf-hash decompression`

SL	TC	Min	Q_1	Q_2	Q_3	Max
4	1	0.68	0.92	0.93	0.98	0.99
4	2	1.33	1.45	1.69	1.76	1.82
4	3	1.89	2.03	2.43	2.55	2.71
4	4	2.29	2.51	3.09	3.42	3.60
16	1	0.76	0.98	0.98	0.99	1.00
16	2	1.43	1.80	1.91	1.94	1.97
16	3	2.41	2.61	2.82	2.87	2.94
16	4	3.14	3.37	3.65	3.84	3.92
32	1	0.77	0.96	0.99	1.00	1.00
32	2	1.77	1.90	1.95	1.98	2.00
32	3	2.71	2.77	2.90	2.94	2.99
32	4	2.93	3.54	3.80	3.90	3.95

Table D.46: `ztfd` speed up compared to minimum single-thread cost

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	9.3	9.7	9.8	13.4	14.4	14.6	16.6	18.1	18.2
Enwiki	8.7	9.1	9.2	17.1	18.4	18.8	20.3	22.0	22.4
Ruwiki	9.3	9.8	9.9	9.7	10.3	10.4	12.0	13.4	13.7
Zhwiki	11.1	11.2	11.2	16.2	17.0	17.1	16.4	16.8	16.7

Table D.47: Relative cost (%) of most expensive kernel in single-threaded execution of `ztf decompression`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	9.1	9.6	9.8	14.0	14.9	14.8	16.6	18.6	18.8
Enwiki	8.7	9.3	9.2	17.0	18.6	18.8	18.8	21.9	22.5
Ruwiki	9.1	9.8	9.9	9.9	10.6	10.6	12.5	14.2	14.1
Zhwiki	11.7	11.6	11.2	16.1	17.3	17.4	14.2	16.2	16.9

Table D.48: Relative cost (%) of most expensive kernel in 2-threaded execution of `ztf decompression`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	9.0	9.7	9.8	14.8	15.3	15.0	16.9	19.5	19.2
Enwiki	8.8	9.1	9.2	17.6	19.2	19.1	19.1	22.6	23.0
Ruwiki	9.1	9.8	9.9	10.1	10.8	10.8	12.5	14.6	14.4
Zhwiki	12.4	11.5	11.5	17.3	17.9	17.8	15.0	16.7	17.2

Table D.49: Relative cost (%) of most expensive kernel in 3-threaded execution of `ztf decompression`

CONFIG	SSE-4.2			AVX2			AVX-512		
File	4	16	32	4	16	32	4	16	32
Arwiki	8.9	9.6	9.7	15.5	15.8	15.4	17.5	20.6	19.8
Enwiki	9.1	9.5	9.4	18.6	20.1	20.0	19.3	24.0	25.9
Ruwiki	9.0	9.8	9.7	10.4	11.3	11.0	13.6	15.2	15.0
Zhwiki	13.4	12.2	11.5	19.1	19.1	18.6	16.2	17.6	18.0

Table D.50: Relative cost (%) of most expensive kernel in 4-threaded execution of `ztf decompression`

D.2 Selected examples of strides per segment diagrams

D.2.1 Case study: `icgrep -always`

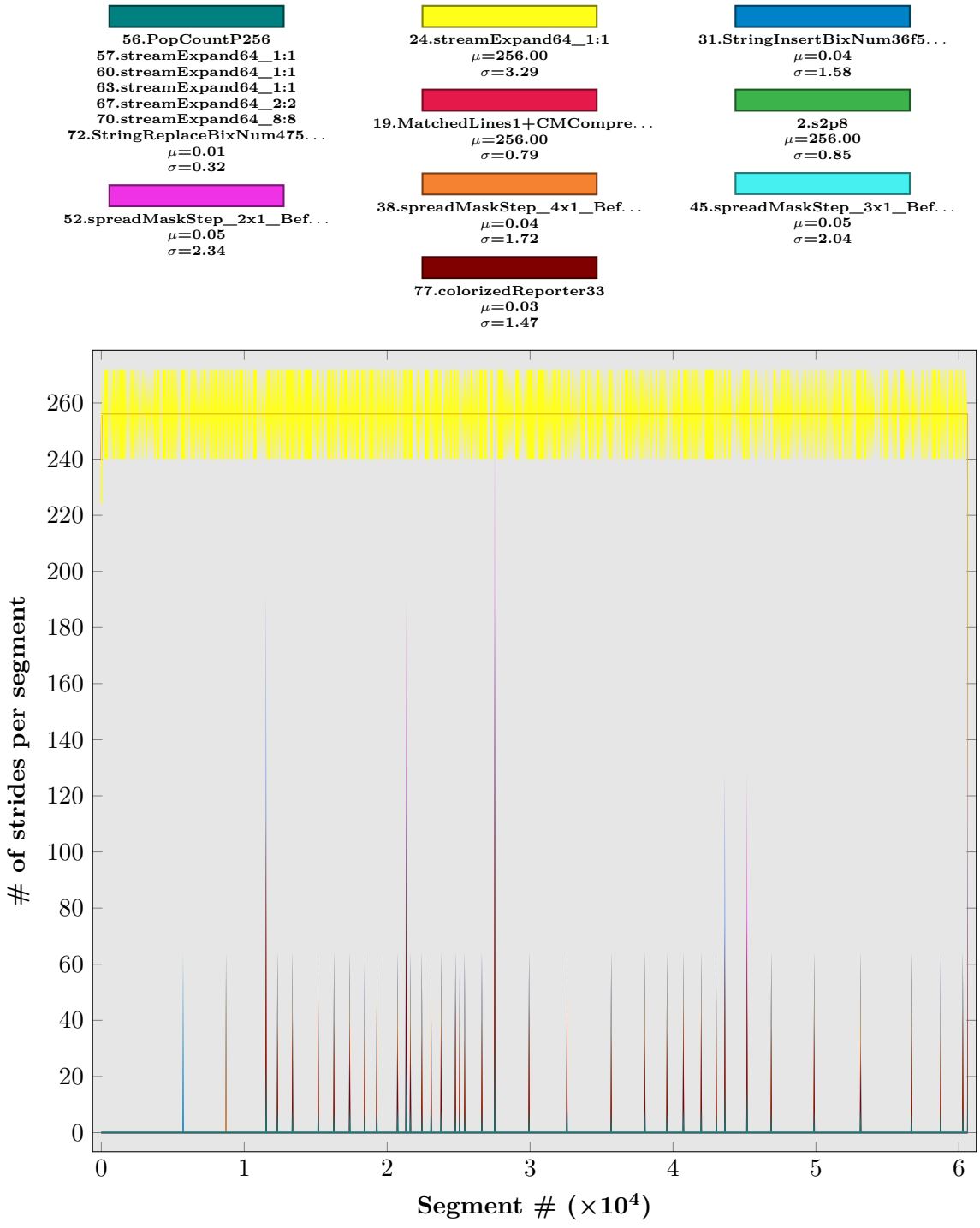


Figure D.1: Min/median/max strides per 50 segments of top 9 most-variable partition roots for `icgrep \p{Greek}` on `Enwiki`

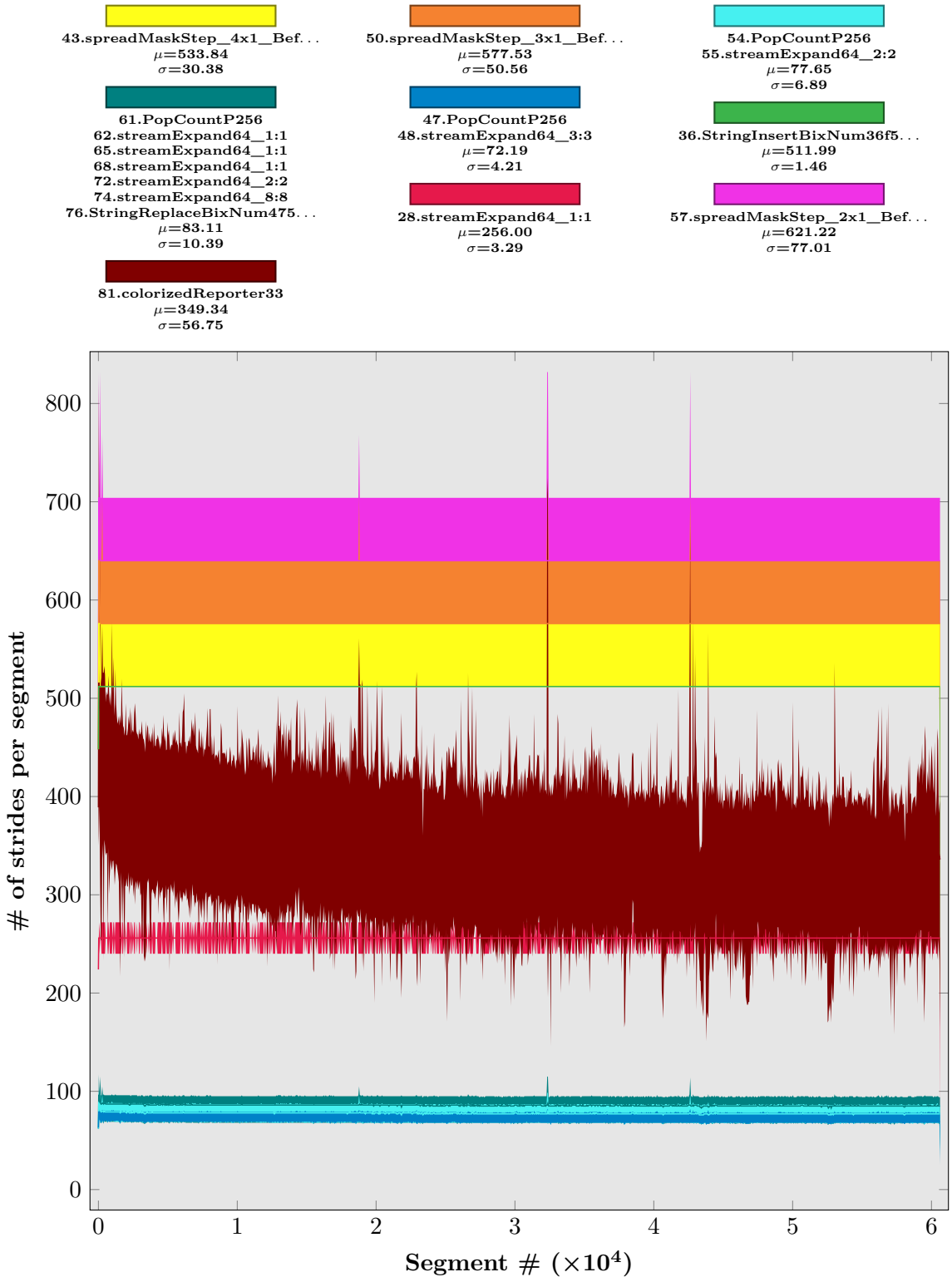


Figure D.2: Min/median/max strides per 50 segments of top 9 most-variable partition roots for `icgrep \X` on `Enwiki`

D.2.2 Case study: gb18030

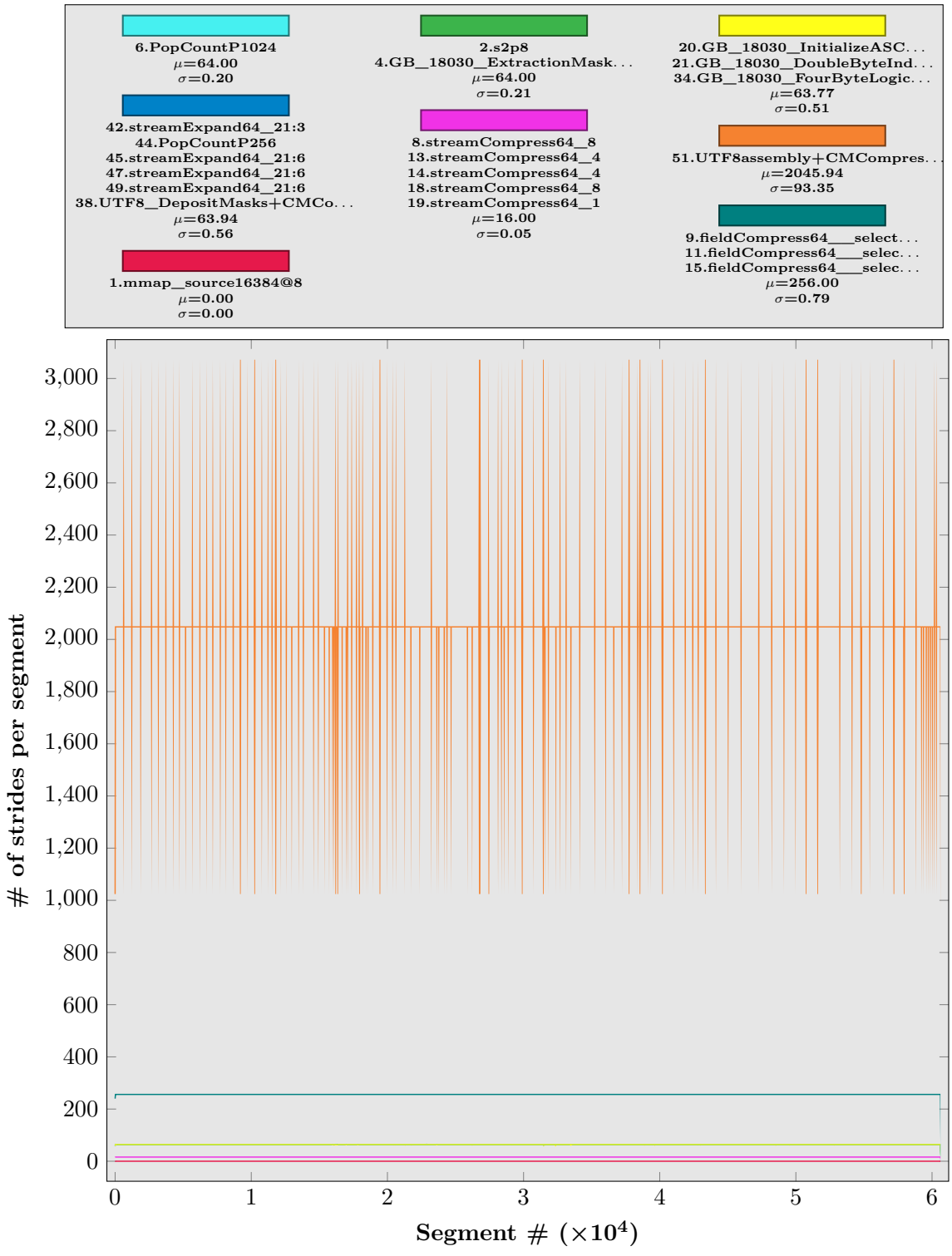


Figure D.3: Min/median/max strides of partition roots per 50 segments of `gb18030` on `Enwiki`

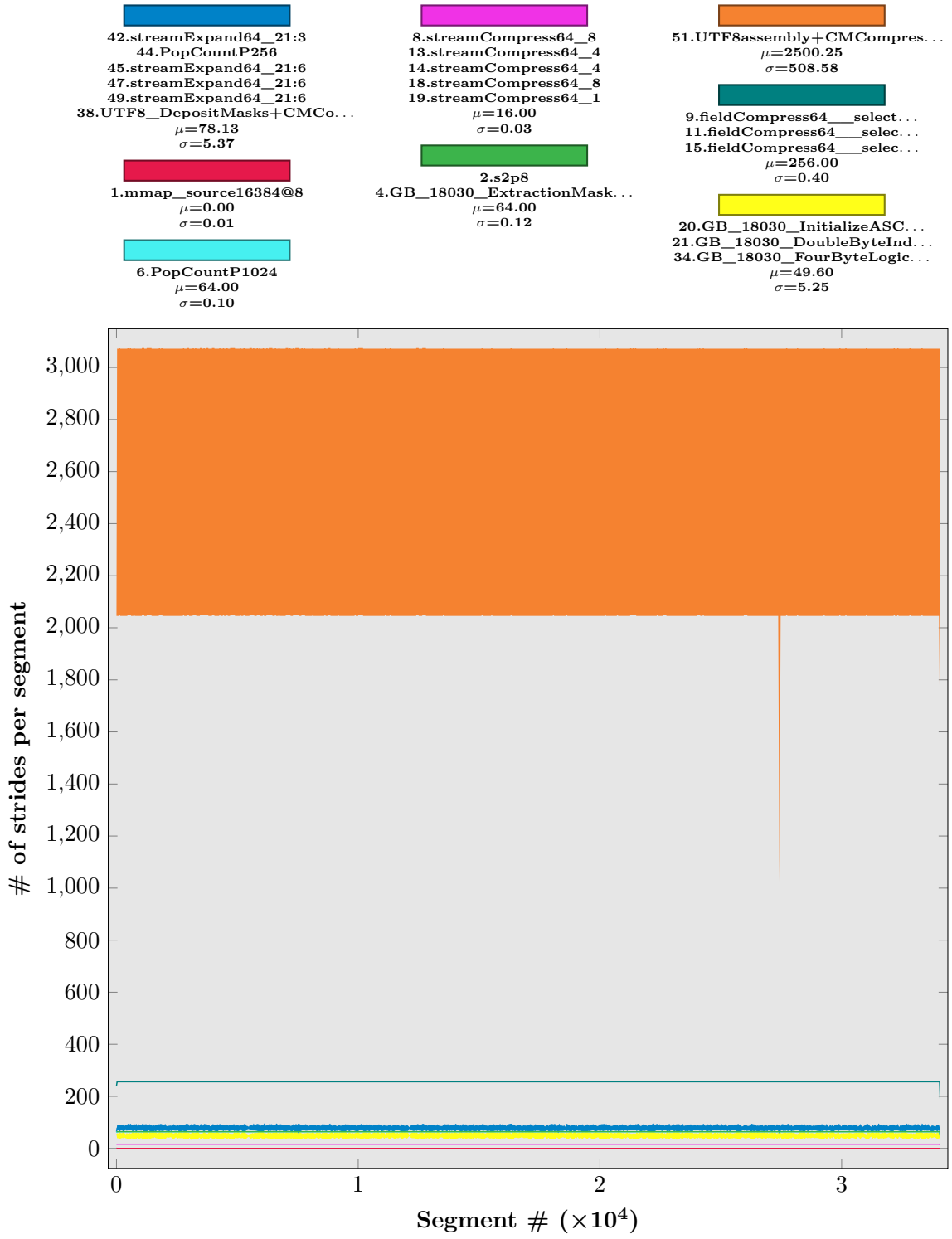


Figure D.4: Min/median/max strides of partition roots per 50 segments of `gb18030` on `Zhwiki`

D.2.3 Case study: `u32u8`

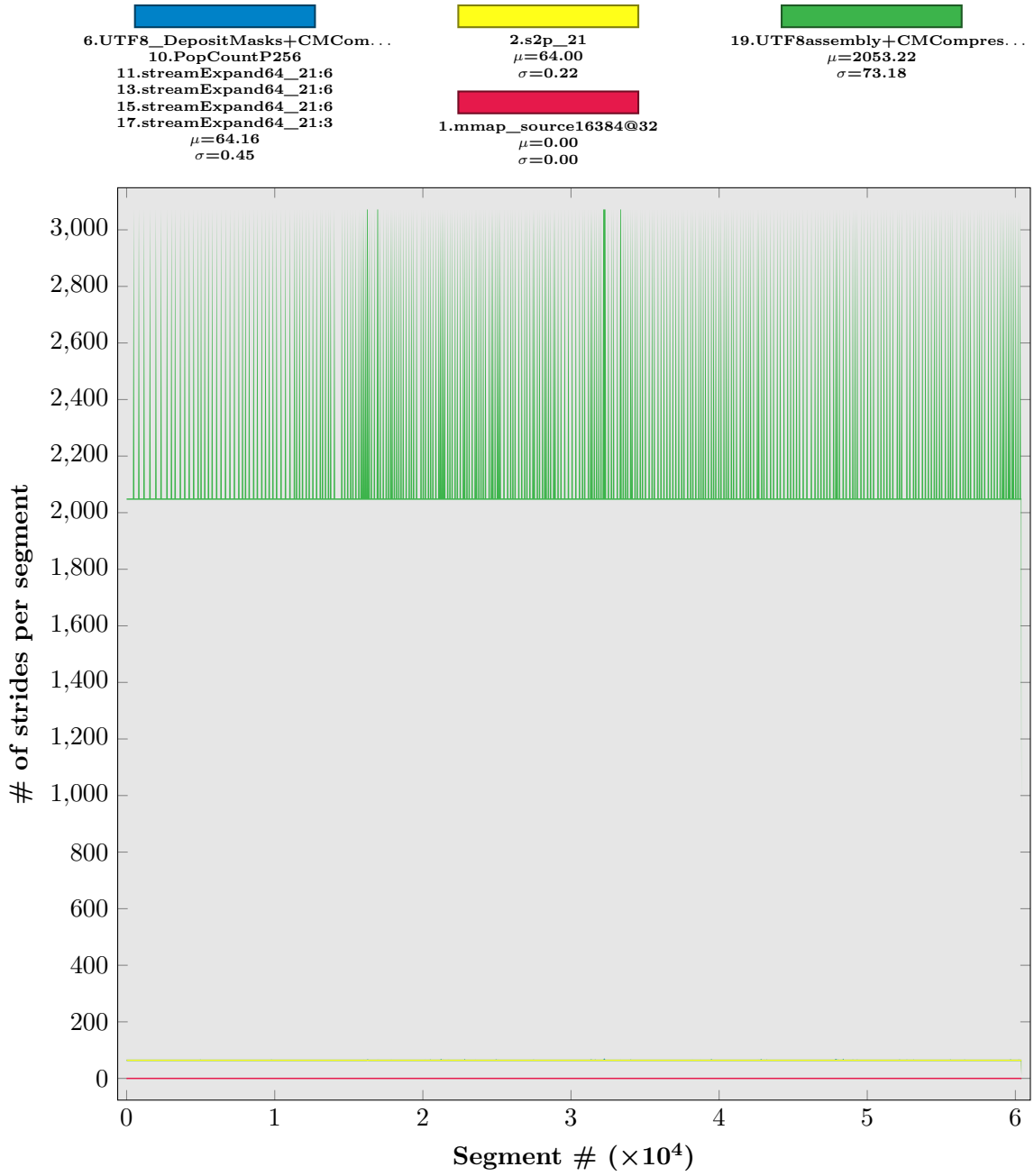


Figure D.5: Min/median/max strides per 50 segments of partition roots for `u32u8` on `Enwiki`

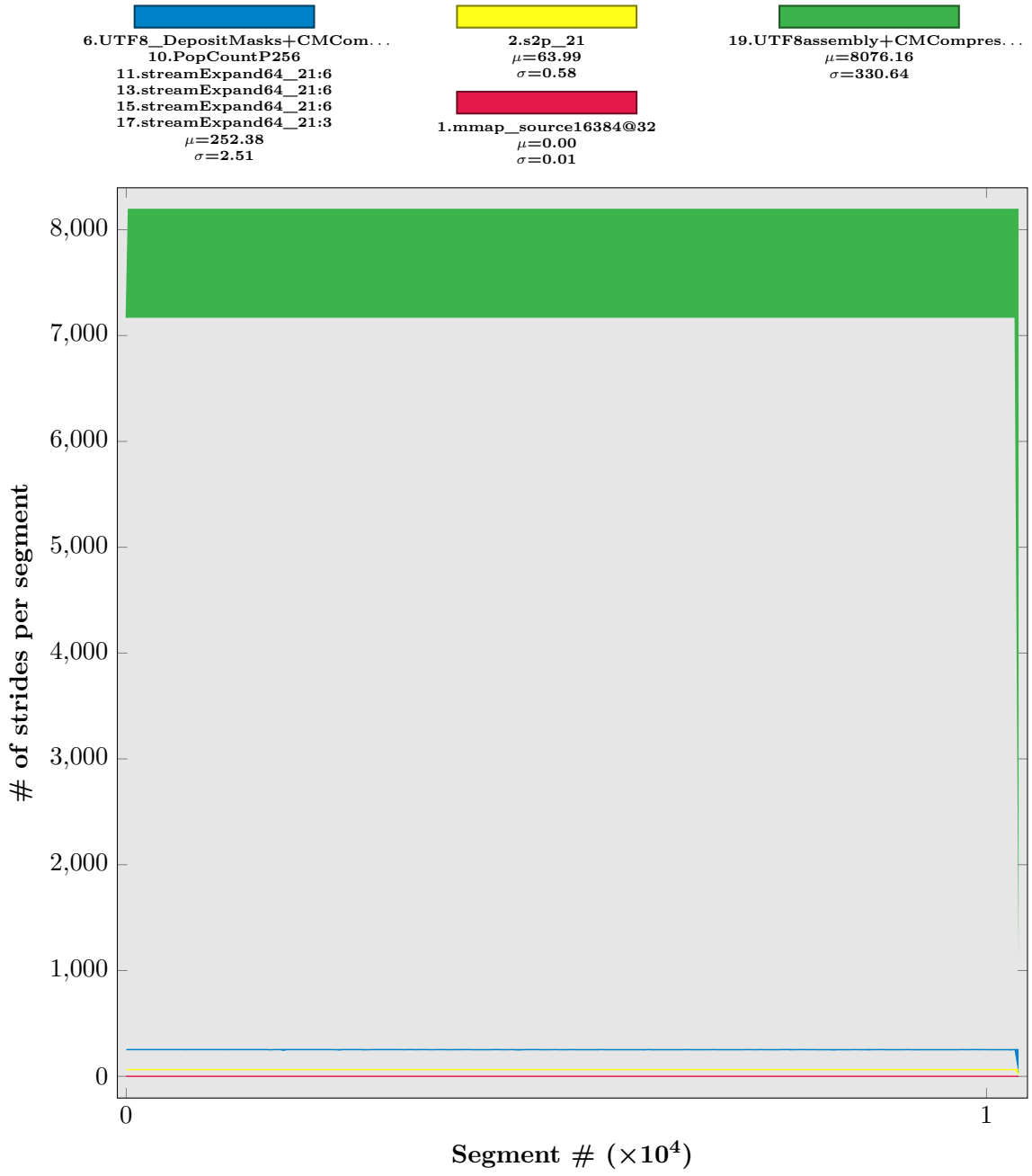


Figure D.6: Min/median/max strides per 50 segments of partition roots for `u32u8` on `Zhwiki`

D.3 Comparison against 2017 version

D.3.1 Case study: `base64`

CONFIG			2017			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	2.45	15.80	6.89	2.21	9.03	3.78	0.16	0.47	2.17
Arwiki	4	2	1.78	38.50	3.86	1.25	6.52	2.23	0.37	2.23	1.14
Arwiki	4	3	2.01	34.70	2.45	1.11	4.90	1.62	0.63	2.08	0.57
Arwiki	4	4	2.29	35.20	2.23	.81	3.23	.95	1.04	2.23	0.90
Arwiki	16	1	2.48	17.60	7.75	2.20	9.31	3.91	0.20	0.58	2.67
Arwiki	16	2	1.55	31.50	3.90	1.16	6.00	2.38	0.27	1.78	1.06
Arwiki	16	3	1.86	28.90	1.94	.80	3.65	1.23	0.74	1.76	0.50
Arwiki	16	4	2.25	31.80	1.85	.67	3.53	1.23	1.10	1.97	0.43
Arwiki	32	1	2.54	27.80	9.33	2.20	10.40	4.31	0.24	1.22	3.50
Arwiki	32	2	1.61	34.20	4.21	1.15	6.08	2.54	0.32	1.96	1.16
Arwiki	32	3	1.90	30.30	2.44	.77	4.24	1.74	0.79	1.82	0.48
Arwiki	32	4	2.24	29.60	1.94	.57	3.12	1.33	1.16	1.85	0.42
Enwiki	4	1	1.69	11.00	4.78	1.53	6.23	2.66	0.16	0.48	2.14
Enwiki	4	2	1.23	26.70	2.50	.87	4.52	1.56	0.37	2.23	0.95
Enwiki	4	3	1.41	24.10	1.61	.77	3.43	1.16	0.65	2.08	0.45
Enwiki	4	4	1.59	25.40	1.44	.49	2.41	.69	1.11	2.32	0.76
Enwiki	16	1	1.71	12.10	5.16	1.52	6.44	2.74	0.19	0.57	2.44
Enwiki	16	2	1.07	21.90	2.75	.81	4.20	1.69	0.26	1.78	1.06
Enwiki	16	3	1.29	19.70	1.31	.57	2.49	.84	0.73	1.74	0.47
Enwiki	16	4	1.56	21.80	1.29	.48	2.45	.86	1.10	1.95	0.43
Enwiki	32	1	1.76	19.70	6.42	1.53	7.10	3.00	0.23	1.27	3.45
Enwiki	32	2	1.11	23.40	2.87	.79	4.18	1.72	0.32	1.93	1.15
Enwiki	32	3	1.34	20.90	1.70	.53	2.94	1.23	0.81	1.81	0.47
Enwiki	32	4	1.57	21.10	1.37	.40	2.15	.91	1.18	1.91	0.47
Ruwiki	4	1	3.02	19.60	8.64	2.73	11.20	4.76	0.17	0.47	2.19
Ruwiki	4	2	2.20	47.10	4.95	1.55	8.02	2.74	0.37	2.21	1.25
Ruwiki	4	3	2.50	43.30	3.06	1.36	6.05	1.99	0.64	2.11	0.61
Ruwiki	4	4	2.80	43.90	2.80	.85	4.24	1.20	1.10	2.25	0.91
Ruwiki	16	1	3.07	21.80	9.84	2.72	11.60	4.92	0.20	0.58	2.78
Ruwiki	16	2	1.92	39.20	4.80	1.44	7.49	2.94	0.27	1.80	1.05
Ruwiki	16	3	2.26	35.50	2.35	.99	4.52	1.52	0.72	1.75	0.47
Ruwiki	16	4	2.73	38.50	2.31	.83	4.32	1.50	1.07	1.93	0.46
Ruwiki	32	1	3.13	33.20	11.40	2.71	12.70	5.25	0.23	1.16	3.50
Ruwiki	32	2	1.99	42.10	5.13	1.42	7.61	3.16	0.32	1.95	1.12
Ruwiki	32	3	2.35	36.60	2.95	.95	5.20	2.13	0.80	1.78	0.46
Ruwiki	32	4	2.74	37.80	2.54	.71	3.85	1.63	1.15	1.92	0.51
Zhwiki	4	1	1.16	7.42	3.21	1.05	4.25	1.80	0.16	0.47	2.07
Zhwiki	4	2	.85	18.20	1.80	.60	3.06	1.04	0.37	2.22	1.12
Zhwiki	4	3	.96	16.50	1.13	.53	2.34	.77	0.64	2.08	0.53
Zhwiki	4	4	1.10	17.50	1.04	.33	1.63	.47	1.13	2.33	0.84
Zhwiki	16	1	1.18	8.06	3.45	1.05	4.34	1.83	0.19	0.55	2.39
Zhwiki	16	2	.73	15.10	1.86	.55	2.82	1.10	0.27	1.80	1.10
Zhwiki	16	3	.88	14.10	.93	.38	1.75	.59	0.73	1.82	0.50
Zhwiki	16	4	1.08	14.70	.86	.33	1.71	.60	1.10	1.91	0.39
Zhwiki	32	1	1.21	13.70	4.49	1.04	4.70	1.92	0.24	1.31	3.77
Zhwiki	32	2	.77	16.00	1.97	.54	2.79	1.14	0.33	1.94	1.22
Zhwiki	32	3	.92	14.00	1.16	.36	1.97	.81	0.81	1.77	0.53
Zhwiki	32	4	1.06	14.60	.97	.29	1.50	.62	1.14	1.92	0.51
								μ	**0.59**	1.67	1.24
								σ	0.36	0.57	0.96

Table D.51: `base64` PAPI comparison between `2017` and `CURR` on `SSE-4.2`

CONFIG			2017			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.18	42.70	24.50	1.08	42.50	27.10	0.07	0.01	-1.81
Arwiki	4	2	1.01	141.00	13.00	.65	29.60	13.80	0.25	7.74	-0.61
Arwiki	4	3	1.50	137.00	8.59	.51	19.90	7.63	0.69	8.18	0.67
Arwiki	4	4	1.57	114.00	6.44	.38	16.70	7.03	0.83	6.79	-0.41
Arwiki	16	1	1.22	123.00	24.50	1.08	42.30	27.70	0.09	5.63	-2.22
Arwiki	16	2	.82	145.00	12.70	.58	25.80	14.10	0.17	8.29	-1.02
Arwiki	16	3	1.24	131.00	8.53	.38	14.40	7.13	0.60	8.17	0.97
Arwiki	16	4	1.35	106.00	6.34	.34	13.60	7.26	0.70	6.46	-0.64
Arwiki	32	1	1.28	276.00	24.50	1.08	51.20	27.80	0.14	15.70	-2.32
Arwiki	32	2	.81	157.00	12.50	.56	28.30	14.10	0.17	8.95	-1.11
Arwiki	32	3	1.18	138.00	8.38	.37	17.00	9.00	0.56	8.44	-0.43
Arwiki	32	4	1.29	111.00	6.27	.30	14.60	7.15	0.69	6.72	-0.61
Enwiki	4	1	.83	28.40	16.60	.76	29.30	18.70	0.07	-0.09	-2.06
Enwiki	4	2	.70	97.00	8.96	.45	20.50	9.52	0.25	7.71	-0.56
Enwiki	4	3	1.04	95.00	5.95	.35	13.80	5.28	0.69	8.19	0.68
Enwiki	4	4	1.09	78.90	4.46	.27	11.80	4.86	0.83	6.77	-0.40
Enwiki	16	1	.85	86.30	16.80	.76	29.00	18.90	0.10	5.77	-2.16
Enwiki	16	2	.57	101.00	8.78	.40	17.80	9.76	0.17	8.35	-0.98
Enwiki	16	3	.86	91.00	5.89	.26	9.93	4.93	0.60	8.16	0.97
Enwiki	16	4	.93	74.60	4.39	.24	9.39	5.02	0.70	6.57	-0.63
Enwiki	32	1	.88	193.00	16.80	.76	34.90	19.00	0.13	16.00	-2.22
Enwiki	32	2	.57	110.00	8.67	.39	19.60	9.82	0.18	9.09	-1.17
Enwiki	32	3	.81	96.30	5.81	.26	11.90	6.23	0.56	8.50	-0.42
Enwiki	32	4	.89	76.70	4.34	.22	10.60	4.96	0.68	6.66	-0.62
Ruwiki	4	1	1.46	53.40	30.20	1.34	52.40	33.40	0.07	0.06	-1.83
Ruwiki	4	2	1.24	173.00	15.90	.80	36.50	17.10	0.25	7.74	-0.66
Ruwiki	4	3	1.84	169.00	10.60	.63	24.50	9.36	0.69	8.20	0.70
Ruwiki	4	4	1.93	140.00	7.93	.47	20.70	8.70	0.83	6.77	-0.44
Ruwiki	16	1	1.50	165.00	30.20	1.33	52.20	34.00	0.10	6.41	-2.16
Ruwiki	16	2	1.03	181.00	15.60	.71	31.80	17.40	0.18	8.44	-1.06
Ruwiki	16	3	1.52	161.00	10.50	.47	17.70	8.78	0.59	8.11	0.98
Ruwiki	16	4	1.65	132.00	7.81	.42	16.70	8.90	0.70	6.50	-0.62
Ruwiki	32	1	1.55	351.00	30.20	1.33	59.60	34.20	0.12	16.50	-2.24
Ruwiki	32	2	.98	188.00	15.40	.69	36.20	17.40	0.17	8.61	-1.09
Ruwiki	32	3	1.44	169.00	10.30	.45	20.90	11.10	0.56	8.38	-0.43
Ruwiki	32	4	1.59	137.00	7.72	.37	18.40	8.81	0.69	6.70	-0.61
Zhwiki	4	1	.56	20.20	11.60	.52	20.20	12.90	0.07	-0.01	-1.85
Zhwiki	4	2	.48	66.60	6.16	.31	14.10	6.59	0.25	7.71	-0.64
Zhwiki	4	3	.71	65.40	4.08	.24	9.47	3.61	0.69	8.21	0.68
Zhwiki	4	4	.74	54.20	3.06	.19	8.02	3.38	0.82	6.78	-0.47
Zhwiki	16	1	.57	49.10	11.70	.52	20.10	13.10	0.09	4.25	-2.16
Zhwiki	16	2	.39	68.00	6.01	.28	12.30	6.72	0.16	8.18	-1.05
Zhwiki	16	3	.58	62.20	4.05	.18	6.81	3.38	0.59	8.13	0.99
Zhwiki	16	4	.64	50.40	3.01	.17	6.56	3.42	0.69	6.44	-0.61
Zhwiki	32	1	.61	130.00	11.60	.51	24.10	13.20	0.14	15.50	-2.32
Zhwiki	32	2	.38	72.50	5.94	.27	13.60	6.70	0.16	8.66	-1.11
Zhwiki	32	3	.56	66.50	3.99	.18	7.96	4.28	0.56	8.59	-0.42
Zhwiki	32	4	.61	52.10	2.98	.15	7.06	3.40	0.68	6.61	-0.62
								μ	**0.41**	7.59	-0.80
								σ	0.27	3.40	0.96

Table D.52: `base64` PAPI comparison between `2017` and `CURR` on `AVX2`

CONFIG			2017			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.07	22.50	22.40	.78	22.60	22.40	0.20	-0.01	0.00
Arwiki	4	2	1.09	61.60	11.20	.55	14.70	11.20	0.37	3.28	0.02
Arwiki	4	3	1.67	61.60	7.46	.43	9.47	6.60	0.86	3.64	0.60
Arwiki	4	4	1.93	49.50	5.60	.43	8.01	5.61	1.05	2.89	-0.01
Arwiki	16	1	1.08	22.50	22.40	.78	22.60	22.40	0.21	-0.01	-0.00
Arwiki	16	2	.76	57.50	11.20	.48	13.00	11.20	0.19	3.11	0.02
Arwiki	16	3	1.67	56.20	7.47	.35	6.92	5.65	0.93	3.44	1.27
Arwiki	16	4	1.93	43.30	5.60	.34	6.84	5.59	1.11	2.55	0.01
Arwiki	32	1	1.04	22.70	22.40	.77	22.60	22.40	0.19	0.01	0.00
Arwiki	32	2	.74	56.60	11.20	.45	12.20	11.20	0.20	3.10	0.02
Arwiki	32	3	1.53	55.40	7.48	.32	7.98	6.99	0.85	3.31	0.34
Arwiki	32	4	1.83	43.20	5.61	.26	6.20	5.60	1.10	2.58	0.01
Enwiki	4	1	.74	15.60	15.50	.54	15.70	15.50	0.20	-0.01	0.00
Enwiki	4	2	.76	42.70	7.76	.38	10.20	7.76	0.38	3.28	-0.00
Enwiki	4	3	1.16	42.70	5.17	.30	6.54	4.58	0.87	3.65	0.60
Enwiki	4	4	1.33	34.00	3.87	.30	5.53	3.90	1.04	2.87	-0.02
Enwiki	16	1	.75	15.60	15.50	.54	15.70	15.50	0.21	-0.01	-0.00
Enwiki	16	2	.53	39.90	7.77	.33	9.01	7.77	0.20	3.11	-0.01
Enwiki	16	3	1.15	39.00	5.17	.24	4.79	3.91	0.92	3.45	1.27
Enwiki	16	4	1.33	30.00	3.88	.24	4.76	3.88	1.10	2.55	-0.01
Enwiki	32	1	.72	15.70	15.50	.54	15.70	15.50	0.19	-0.00	0.00
Enwiki	32	2	.51	39.20	7.78	.30	8.45	7.73	0.21	3.10	0.05
Enwiki	32	3	1.05	37.50	5.18	.22	5.53	4.84	0.83	3.23	0.34
Enwiki	32	4	1.31	30.10	3.89	.17	4.30	3.88	1.15	2.60	0.01
Ruwiki	4	1	1.33	27.80	27.60	.96	27.90	27.60	0.21	-0.01	0.00
Ruwiki	4	2	1.34	75.80	13.80	.69	18.10	13.90	0.37	3.27	-0.02
Ruwiki	4	3	2.04	76.00	9.20	.53	11.70	8.12	0.86	3.64	0.61
Ruwiki	4	4	2.37	60.80	6.90	.53	9.88	6.86	1.04	2.89	0.02
Ruwiki	16	1	1.34	27.80	27.60	.96	27.90	27.60	0.21	-0.01	-0.00
Ruwiki	16	2	.94	71.00	13.80	.59	16.10	13.70	0.20	3.11	0.04
Ruwiki	16	3	2.03	69.30	9.20	.42	8.50	6.96	0.91	3.44	1.27
Ruwiki	16	4	2.35	53.80	6.90	.42	8.47	6.91	1.09	2.57	-0.00
Ruwiki	32	1	1.29	28.00	27.60	.95	27.90	27.60	0.19	0.01	0.00
Ruwiki	32	2	.90	69.70	13.80	.55	15.00	13.80	0.20	3.10	0.04
Ruwiki	32	3	1.86	66.70	9.22	.39	9.84	8.61	0.83	3.22	0.34
Ruwiki	32	4	2.28	53.20	6.92	.32	7.64	6.90	1.11	2.58	0.01
Zhwiki	4	1	.51	10.70	10.60	.37	10.80	10.60	0.20	-0.01	0.00
Zhwiki	4	2	.51	29.30	5.34	.27	6.96	5.35	0.36	3.28	-0.03
Zhwiki	4	3	.80	29.20	3.55	.20	4.46	3.14	0.88	3.64	0.60
Zhwiki	4	4	.93	23.60	2.66	.21	3.77	2.66	1.06	2.90	-0.01
Zhwiki	16	1	.51	10.70	10.60	.37	10.80	10.60	0.21	-0.00	-0.00
Zhwiki	16	2	.36	27.40	5.33	.23	6.18	5.33	0.19	3.11	0.00
Zhwiki	16	3	.79	26.80	3.55	.17	3.28	2.69	0.92	3.45	1.27
Zhwiki	16	4	.92	20.60	2.66	.16	3.25	2.67	1.10	2.54	-0.01
Zhwiki	32	1	.49	10.80	10.60	.37	10.80	10.60	0.19	0.01	0.00
Zhwiki	32	2	.35	26.90	5.34	.21	5.80	5.33	0.20	3.10	0.02
Zhwiki	32	3	.75	26.50	3.55	.15	3.79	3.33	0.88	3.33	0.34
Zhwiki	32	4	.92	20.90	2.67	.12	2.95	2.66	1.17	2.63	0.01
								μ	**0.61**	2.32	0.19
								σ	0.39	1.38	0.37

Table D.53: `base64` PAPI comparison between `2017` and `CURR` on `AVX-512`

D.3.2 Case study: `editd`

CONFIG			2017			CURR			NORM. DIFFERENCE		
DIST	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
1	4	1	20.90	205.00	1.07	19.20	285.00	1.21	7.00	-31.60	-0.57
1	4	3	8.82	91.30	.32	6.43	96.70	.40	9.39	-2.11	-0.32
1	4	4	6.81	70.30	.46	4.92	74.30	.30	7.46	-1.56	0.62
1	16	1	17.20	117.00	.99	17.00	217.00	1.01	0.78	-39.10	-0.10
1	16	3	7.32	46.80	.31	5.77	73.40	.34	6.12	-10.50	-0.10
1	16	4	5.47	40.40	.25	4.31	55.60	.26	4.58	-5.96	-0.05
1	32	1	16.60	118.00	1.00	16.70	214.00	.97	-0.47	-37.90	0.12
1	32	3	7.22	43.80	.31	5.56	71.00	.33	6.52	-10.70	-0.09
1	32	4	5.20	35.50	.24	4.20	53.40	.31	3.93	-7.05	-0.28
2	4	1	35.40	328.00	1.03	33.00	456.00	1.18	9.26	-50.10	-0.59
2	4	3	14.40	150.00	.31	11.10	153.00	.40	13.00	-1.45	-0.33
2	4	4	11.20	122.00	.46	8.48	118.00	.29	10.50	1.63	0.67
2	16	1	29.50	200.00	.99	30.00	396.00	1.01	-1.79	-77.10	-0.09
2	16	3	12.40	79.60	.31	10.10	131.00	.34	9.21	-20.40	-0.14
2	16	4	9.35	67.60	.25	7.50	99.80	.30	7.27	-12.70	-0.21
2	32	1	28.50	188.00	.98	29.70	384.00	1.00	-4.80	-77.00	-0.06
2	32	3	11.60	68.90	.30	9.81	127.00	.45	7.09	-22.80	-0.57
2	32	4	8.99	59.90	.27	7.32	95.00	1.11	6.55	-13.80	-3.31
4	4	1	73.00	639.00	1.09	70.90	597.00	1.24	8.35	16.70	-0.58
4	4	3	28.90	248.00	.34	25.40	204.00	.41	13.60	17.50	-0.27
4	4	4	22.00	214.00	.48	18.30	154.00	.31	14.80	23.90	0.70
4	16	1	62.90	371.00	1.03	66.00	517.00	1.04	-12.30	-57.30	-0.05
4	16	3	25.40	134.00	.33	22.00	174.00	.73	13.30	-15.60	-1.58
4	16	4	20.10	113.00	.28	16.50	130.00	1.88	14.10	-6.64	-6.31
4	32	1	60.90	322.00	1.01	64.70	501.00	1.07	-15.10	-70.50	-0.23
4	32	3	23.80	114.00	.51	22.20	163.00	12.70	6.35	-19.20	-48.00
4	32	4	18.70	94.00	1.18	17.40	121.00	23.40	5.11	-10.60	-87.40
8	4	1	199.00	1020.00	17.30	178.00	915.00	17.90	80.40	43.30	-2.08
8	4	3	233.00	370.00	.95	69.30	326.00	9.67	642.00	17.10	-34.30
8	4	4				58.70	248.00	12.30			
8	16	1	182.00	663.00	17.50	172.00	813.00	18.90	41.10	-59.10	-5.41
8	16	3	155.00	250.00	15.10	66.70	290.00	43.60	349.00	-16.00	-112.00
8	16	4				143.00	203.00	51.20			
8	32	1	179.00	599.00	18.60	172.00	785.00	33.30	29.00	-73.30	-57.60
8	32	3	156.00	225.00	40.30	157.00	252.00	107.00	-3.86	-10.30	-261.00
8	32	4				152.00	180.00	104.00			
								μ	**39.00**	-19.40	-18.90
								σ	122.00	29.70	50.40

Table D.54: `editd` PAPI comparison between `2017` and `CURR` on `SSE-4.2`

CONFIG			2017			CURR			NORM. DIFFERENCE		
DIST	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
1	4	1	13.50	1140.00	5.32	12.00	1180.00	5.51	5.75	-16.70	-0.75
1	4	3	5.25	454.00	1.92	4.08	406.00	1.91	4.62	18.80	0.05
1	4	4	3.63	344.00	1.41	3.09	306.00	1.44	2.13	15.10	-0.12
1	16	1	12.20	583.00	5.58	11.10	558.00	5.53	4.41	9.80	0.20
1	16	3	4.62	208.00	1.93	3.71	188.00	1.80	3.61	7.89	0.52
1	16	4	3.11	158.00	1.42	2.80	143.00	1.37	1.22	5.94	0.18
1	32	1	12.00	511.00	5.68	10.90	533.00	5.41	4.17	-8.70	1.06
1	32	3	4.53	175.00	1.98	3.65	177.00	1.77	3.46	-0.74	0.84
1	32	4	3.03	131.00	1.59	2.75	132.00	1.36	1.09	-0.49	0.91
2	4	1	21.00	1680.00	5.41	18.80	1670.00	5.42	8.92	3.77	-0.04
2	4	3	7.80	629.00	1.88	6.34	570.00	1.86	5.77	23.10	0.09
2	4	4	5.56	471.00	1.38	4.78	429.00	1.40	3.04	16.40	-0.08
2	16	1	19.70	863.00	5.67	17.80	1010.00	5.57	7.66	-59.20	0.37
2	16	3	7.13	299.00	1.97	5.96	343.00	1.83	4.61	-17.10	0.57
2	16	4	4.90	225.00	1.48	4.49	259.00	1.41	1.62	-13.20	0.27
2	32	1	19.50	729.00	5.70	17.50	819.00	5.40	7.86	-35.20	1.19
2	32	3	7.05	252.00	2.92	5.85	276.00	1.80	4.71	-9.42	4.41
2	32	4	4.95	188.00	5.78	4.41	207.00	1.93	2.13	-7.69	15.10
4	4	1	44.20	2820.00	5.61	39.40	2490.00	5.52	19.00	127.00	0.34
4	4	3	15.70	1010.00	1.97	13.20	851.00	1.87	9.61	61.10	0.43
4	4	4	11.40	756.00	1.47	9.96	638.00	1.42	5.64	46.50	0.23
4	16	1	42.50	1460.00	5.85	37.90	1570.00	5.54	18.10	-44.60	1.20
4	16	3	14.90	501.00	2.82	12.70	529.00	2.38	8.73	-11.00	1.76
4	16	4	10.70	373.00	6.25	9.55	398.00	4.81	4.38	-9.93	5.67
4	32	1	42.20	1180.00	5.99	37.60	1370.00	5.54	18.00	-75.80	1.77
4	32	3	15.00	408.00	42.40	12.70	462.00	23.60	8.95	-21.50	74.20
4	32	4	10.90	308.00	70.20	9.86	343.00	72.00	4.16	-14.00	-6.83
8	4	1	132.00	5410.00	170.00	105.00	4960.00	169.00	110.00	179.00	4.22
8	4	3	51.90	1930.00	89.60	41.30	1730.00	89.70	41.70	79.00	-0.34
8	4	4				32.80	1350.00	109.00			
8	16	1	131.00	2800.00	176.00	104.00	3030.00	171.00	107.00	-91.80	20.50
8	16	3	50.30	1080.00	290.00	40.10	1130.00	267.00	40.30	-17.70	89.80
8	16	4				34.70	765.00	260.00			
8	32	1	132.00	2360.00	276.00	103.00	2680.00	201.00	112.00	-126.00	295.00
8	32	3	53.20	825.00	411.00	44.00	865.00	425.00	36.20	-15.60	-55.40
8	32	4				36.40	612.00	330.00			
								μ	**18.80**	-0.08	13.90
								σ	30.60	55.30	54.50

Table D.55: `editd` PAPI comparison between `2017` and `CURR` on `AVX2`

CONFIG			2017			CURR			NORM. DIFFERENCE		
DIST	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
1	4	1	9.67	8.98	4.02	7.01	55.90	4.00	10.50	-18.40	0.08
1	4	3	4.55	88.20	1.37	2.81	26.10	1.41	6.87	24.40	-0.18
1	4	4	3.17	67.80	1.02	2.13	21.40	1.05	4.07	18.20	-0.11
1	16	1	9.14	43.50	4.02	6.09	31.50	4.00	12.00	4.71	0.11
1	16	3	3.80	45.30	1.50	2.13	14.10	1.40	6.57	12.30	0.38
1	16	4	2.54	34.90	1.13	1.61	10.80	1.05	3.67	9.46	0.30
1	32	1	9.18	129.00	4.02	6.07	88.00	4.01	12.20	16.00	0.07
1	32	3	3.63	48.50	1.62	2.06	32.20	1.36	6.17	6.37	1.01
1	32	4	2.38	36.40	1.21	1.57	24.40	1.03	3.16	4.69	0.70
2	4	1	15.60	102.00	4.03	11.10	157.00	4.01	17.70	-21.50	0.08
2	4	3	6.72	144.00	1.90	4.21	64.50	1.47	9.88	31.40	1.70
2	4	4	4.82	98.60	1.22	3.20	49.30	1.09	6.36	19.40	0.53
2	16	1	14.60	189.00	4.06	10.10	198.00	4.01	17.80	-3.27	0.18
2	16	3	5.57	83.10	1.84	3.44	71.30	1.38	8.38	4.65	1.83
2	16	4	3.77	60.70	1.28	2.60	51.90	1.04	4.60	3.48	0.96
2	32	1	14.90	296.00	4.04	10.10	264.00	4.01	18.80	12.50	0.12
2	32	3	5.54	105.00	1.50	3.38	89.30	1.36	8.48	6.29	0.53
2	32	4	3.85	77.90	1.13	2.56	67.20	1.05	5.08	4.18	0.34
4	4	1	33.20	556.00	4.21	22.80	405.00	4.12	41.00	59.20	0.34
4	4	3	12.60	293.00	2.50	8.15	162.00	1.71	17.40	51.70	3.12
4	4	4	9.20	214.00	1.71	6.15	122.00	1.30	12.00	36.10	1.62
4	16	1	31.60	591.00	4.17	21.90	532.00	4.13	37.90	23.00	0.14
4	16	3	11.30	204.00	1.89	7.39	182.00	1.52	15.30	8.54	1.43
4	16	4	8.10	152.00	1.29	5.58	137.00	1.23	9.94	5.82	0.21
4	32	1	31.10	591.00	4.23	22.20	564.00	4.19	35.10	10.60	0.16
4	32	3	11.10	204.00	5.44	7.48	191.00	3.97	14.20	5.05	5.78
4	32	4	7.98	153.00	9.79	5.74	144.00	8.07	8.80	3.65	6.78
8	4	1	101.00	2000.00	140.00	61.50	1670.00	139.00	156.00	129.00	2.49
8	4	3	44.50	770.00	84.90	30.50	607.00	73.00	54.80	64.00	47.00
8	4	4				25.20	492.00	86.50			
8	16	1	96.70	1360.00	143.00	59.40	1320.00	142.00	147.00	14.60	4.04
8	16	3	40.10	523.00	120.00	27.20	500.00	121.00	50.80	8.95	-1.54
8	16	4				30.50	335.00	63.20			
8	32	1	95.70	1230.00	167.00	59.50	1190.00	160.00	142.00	15.60	26.70
8	32	3	47.10	408.00	183.00	35.00	390.00	158.00	47.60	7.25	99.20
8	32	4				33.80	271.00	133.00			
								μ	**28.90**	17.50	6.25
								σ	40.40	26.70	18.80

Table D.56: `editd` PAPI comparison between `2017` and `CURR` on `AVX-512`

D.3.3 Case study: u8u16

CONFIG			2017			CURR′			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	4.24	35.40	1.08	2.73	10.20	.91	1.06	1.76	0.12
Arwiki	4	3	2.97	33.10	.36	2.06	9.60	.73	0.63	1.64	-0.26
Arwiki	16	2	3.98	16.40	.57	2.63	7.72	.97	0.94	0.60	-0.28
Arwiki	16	3	2.66	18.00	.42	1.84	6.58	.85	0.58	0.80	-0.30
Arwiki	32	2	3.86	14.20	.65	2.61	11.30	.96	0.87	0.21	-0.22
Arwiki	32	3	2.50	15.80	.49	1.65	6.26	.78	0.60	0.67	-0.20
Enwiki	4	2	3.05	29.10	.68	1.93	7.76	.62	1.13	2.15	0.06
Enwiki	4	3	2.15	23.90	.24	1.47	7.86	.49	0.68	1.62	-0.25
Enwiki	16	2	2.79	12.40	.33	1.86	7.35	.72	0.94	0.51	-0.39
Enwiki	16	3	1.87	13.40	.27	1.38	7.28	.55	0.49	0.61	-0.28
Enwiki	32	2	2.72	12.00	.42	1.85	11.00	.74	0.88	0.10	-0.32
Enwiki	32	3	1.83	12.20	.30	1.23	7.09	.58	0.60	0.52	-0.29
Ruwiki	4	2	5.43	45.50	1.38	3.52	12.80	1.13	1.08	1.85	0.14
Ruwiki	4	3	3.79	38.10	.47	2.71	12.80	.89	0.61	1.43	-0.24
Ruwiki	16	2	4.95	18.50	.70	3.41	10.30	1.14	0.87	0.47	-0.25
Ruwiki	16	3	3.29	20.00	.54	2.50	9.34	1.05	0.45	0.60	-0.29
Ruwiki	32	2	4.84	16.80	.82	3.28	14.30	1.33	0.88	0.14	-0.29
Ruwiki	32	3	3.09	17.90	.62	2.09	6.75	.88	0.57	0.63	-0.15
Zhwiki	4	2	2.13	17.10	.51	1.43	5.30	.43	1.02	1.73	0.11
Zhwiki	4	3	1.45	15.50	.17	1.13	5.51	.35	0.46	1.47	-0.26
Zhwiki	16	2	1.96	7.38	.28	1.41	4.48	.47	0.81	0.43	-0.28
Zhwiki	16	3	1.27	7.40	.21	1.04	4.20	.39	0.33	0.47	-0.26
Zhwiki	32	2	1.91	6.44	.30	1.45	5.55	.42	0.69	0.13	-0.18
Zhwiki	32	3	1.27	6.60	.25	.84	2.66	.33	0.63	0.58	-0.12
								μ	**0.74**	0.88	-0.20
								σ	0.22	0.62	0.14

Table D.57: u8u16 PAPI comparison between 2017 and CURR′ on SSE-4.2

CONFIG			2017			CURR′			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.56	217.00	15.20	1.91	67.80	15.80	0.45	10.40	-0.44
Arwiki	4	3	1.80	175.00	9.87	1.59	57.20	10.40	0.15	8.25	-0.40
Arwiki	16	2	2.31	200.00	15.70	1.74	61.70	16.00	0.39	9.65	-0.20
Arwiki	16	3	1.50	148.00	10.20	1.38	43.10	10.40	0.08	7.34	-0.14
Arwiki	32	2	2.26	212.00	15.90	1.74	95.60	15.80	0.36	8.14	0.05
Arwiki	32	3	1.37	147.00	10.20	1.36	67.80	10.30	0.01	5.50	-0.08
Enwiki	4	2	1.84	177.00	10.50	1.38	56.30	10.90	0.46	12.20	-0.47
Enwiki	4	3	1.27	133.00	6.83	1.14	51.80	7.24	0.13	8.17	-0.41
Enwiki	16	2	1.63	158.00	10.90	1.25	56.90	11.00	0.38	10.20	-0.17
Enwiki	16	3	1.06	113.00	7.08	1.04	49.80	7.19	0.02	6.37	-0.10
Enwiki	32	2	1.60	160.00	11.00	1.24	88.50	10.90	0.36	7.21	0.15
Enwiki	32	3	.98	110.00	7.11	1.02	65.60	7.13	-0.04	4.47	-0.02
Ruwiki	4	2	3.25	280.00	18.70	2.45	79.90	19.50	0.46	11.30	-0.44
Ruwiki	4	3	2.23	207.00	12.10	2.02	72.20	12.80	0.12	7.61	-0.41
Ruwiki	16	2	2.90	250.00	19.40	2.23	73.80	19.60	0.38	10.00	-0.13
Ruwiki	16	3	1.86	177.00	12.60	1.70	57.70	12.80	0.09	6.76	-0.10
Ruwiki	32	2	2.86	256.00	19.60	2.20	106.00	19.50	0.37	8.53	0.04
Ruwiki	32	3	1.71	176.00	12.60	1.66	71.70	12.80	0.03	5.94	-0.07
Zhwiki	4	2	1.28	107.00	7.19	.96	31.20	7.51	0.47	11.10	-0.47
Zhwiki	4	3	.89	83.50	4.72	.82	29.30	4.97	0.11	7.95	-0.37
Zhwiki	16	2	1.15	96.50	7.44	.88	29.50	7.57	0.40	9.84	-0.18
Zhwiki	16	3	.73	68.70	4.82	.66	21.20	4.95	0.10	6.98	-0.18
Zhwiki	32	2	1.12	100.00	7.54	.87	44.00	7.51	0.37	8.22	0.05
Zhwiki	32	3	.67	69.00	4.88	.64	28.10	4.92	0.05	6.02	-0.06
								μ	**0.24**	8.26	-0.19
								σ	0.17	1.95	0.19

Table D.58: `u8u16` PAPI comparison between `2017` and `CURR′` on `AVX2`

CONFIG			2017			CURR′			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.59	107.00	10.90	1.36	31.10	11.20	0.86	5.27	-0.22
Arwiki	4	3	1.98	85.20	7.48	1.09	24.90	7.47	0.61	4.21	0.01
Arwiki	16	2	2.21	86.00	11.30	1.21	20.30	11.20	0.70	4.59	0.04
Arwiki	16	3	1.56	65.20	7.52	.99	19.40	7.47	0.40	3.20	0.04
Arwiki	32	2	2.14	81.70	11.50	1.11	19.40	11.30	0.71	4.35	0.18
Arwiki	32	3	1.48	60.60	7.84	.79	16.50	7.48	0.48	3.08	0.26
Enwiki	4	2	1.81	84.70	7.80	.95	19.90	7.76	0.87	6.53	0.05
Enwiki	4	3	1.39	64.20	5.18	.74	21.30	5.18	0.66	4.32	0.00
Enwiki	16	2	1.57	71.90	7.79	.86	18.90	7.76	0.71	5.34	0.03
Enwiki	16	3	1.09	51.20	5.20	.71	19.70	5.17	0.38	3.18	0.03
Enwiki	32	2	1.51	69.20	8.06	.80	17.50	7.81	0.71	5.21	0.24
Enwiki	32	3	1.01	48.10	5.36	.69	18.20	5.20	0.33	3.02	0.16
Ruwiki	4	2	3.29	136.00	13.40	1.74	37.50	13.80	0.88	5.58	-0.22
Ruwiki	4	3	2.49	103.00	9.22	1.41	30.00	9.20	0.61	4.15	0.01
Ruwiki	16	2	2.79	112.00	13.90	1.53	26.40	13.80	0.72	4.87	0.04
Ruwiki	16	3	1.92	77.10	9.26	1.25	24.70	9.20	0.38	2.97	0.03
Ruwiki	32	2	2.69	107.00	14.40	1.41	28.10	13.80	0.73	4.49	0.34
Ruwiki	32	3	1.76	71.10	9.61	.95	21.90	9.22	0.46	2.79	0.22
Zhwiki	4	2	1.26	52.20	5.36	.68	14.50	5.33	0.85	5.54	0.04
Zhwiki	4	3	.95	40.50	3.56	.54	11.40	3.55	0.60	4.28	0.01
Zhwiki	16	2	1.08	42.80	5.36	.60	10.60	5.33	0.71	4.72	0.04
Zhwiki	16	3	.74	30.70	3.58	.48	9.15	3.55	0.38	3.16	0.04
Zhwiki	32	2	1.04	41.00	5.54	.55	10.30	5.37	0.71	4.50	0.25
Zhwiki	32	3	.68	28.10	3.71	.37	8.11	3.56	0.45	2.93	0.21
								μ	**0.62**	4.26	0.08
								σ	0.17	1.01	0.13

Table D.59: `u8u16` PAPI comparison between `2017` and `CURR′` on `AVX-512`

D.4 Comparison against 2020 version

D.4.1 Case study: `base64`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	2.51	8.66	3.42	2.21	9.03	3.78	0.21	-0.03	-0.25
Arwiki	4	2	1.44	26.60	1.81	1.25	6.52	2.23	0.13	1.40	-0.29
Arwiki	4	3	1.19	16.90	1.20	1.11	4.90	1.62	0.06	0.84	-0.30
Arwiki	4	4	1.21	10.20	.59	.81	3.23	.95	0.28	0.48	-0.25
Arwiki	16	1	2.31	10.20	4.50	2.20	9.31	3.91	0.08	0.06	0.41
Arwiki	16	2	1.31	14.10	1.99	1.16	6.00	2.38	0.10	0.56	-0.27
Arwiki	16	3	1.05	11.50	1.22	.80	3.65	1.23	0.17	0.55	-0.00
Arwiki	16	4	.96	9.15	.61	.67	3.53	1.23	0.20	0.39	-0.43
Arwiki	32	1	2.28	13.60	4.78	2.20	10.40	4.31	0.06	0.23	0.33
Arwiki	32	2	1.21	10.70	2.11	1.15	6.08	2.54	0.05	0.32	-0.29
Arwiki	32	3	.92	8.38	1.32	.77	4.24	1.74	0.10	0.29	-0.29
Arwiki	32	4	1.02	6.83	.75	.57	3.12	1.33	0.31	0.26	-0.40
Enwiki	4	1	1.74	5.92	2.36	1.53	6.23	2.66	0.21	-0.03	-0.30
Enwiki	4	2	1.05	15.70	1.10	.87	4.52	1.56	0.18	1.13	-0.46
Enwiki	4	3	.82	12.10	.86	.77	3.43	1.16	0.05	0.87	-0.30
Enwiki	4	4	.81	7.19	.43	.49	2.41	.69	0.32	0.48	-0.26
Enwiki	16	1	1.59	6.69	2.88	1.52	6.44	2.74	0.07	0.02	0.14
Enwiki	16	2	.90	10.10	1.43	.81	4.20	1.69	0.09	0.59	-0.27
Enwiki	16	3	.70	8.45	.94	.57	2.49	.84	0.14	0.60	0.10
Enwiki	16	4	.69	5.44	.35	.48	2.45	.86	0.22	0.30	-0.52
Enwiki	32	1	1.57	9.14	3.07	1.53	7.10	3.00	0.04	0.20	0.07
Enwiki	32	2	.87	7.16	1.42	.79	4.18	1.72	0.07	0.30	-0.30
Enwiki	32	3	.62	5.85	.91	.53	2.94	1.23	0.09	0.29	-0.33
Enwiki	32	4	.67	4.97	.64	.40	2.15	.91	0.27	0.28	-0.27
Ruwiki	4	1	3.10	10.60	4.24	2.73	11.20	4.76	0.21	-0.04	-0.30
Ruwiki	4	2	1.78	32.50	2.10	1.55	8.02	2.74	0.13	1.39	-0.36
Ruwiki	4	3	1.47	21.20	1.50	1.36	6.05	1.99	0.06	0.86	-0.27
Ruwiki	4	4	1.43	14.10	.80	.85	4.24	1.20	0.33	0.56	-0.23
Ruwiki	16	1	2.84	12.20	5.22	2.72	11.60	4.92	0.07	0.04	0.17
Ruwiki	16	2	1.60	17.20	2.37	1.44	7.49	2.94	0.09	0.55	-0.33
Ruwiki	16	3	1.27	14.30	1.54	.99	4.52	1.52	0.16	0.55	0.01
Ruwiki	16	4	1.25	9.73	.58	.83	4.32	1.50	0.24	0.31	-0.52
Ruwiki	32	1	2.79	16.80	5.54	2.71	12.70	5.25	0.04	0.24	0.17
Ruwiki	32	2	1.60	12.30	2.28	1.42	7.61	3.16	0.10	0.27	-0.50
Ruwiki	32	3	1.15	10.30	1.63	.95	5.20	2.13	0.12	0.29	-0.29
Ruwiki	32	4	1.26	8.49	.98	.71	3.85	1.63	0.31	0.26	-0.37
Zhwiki	4	1	1.19	4.03	1.60	1.05	4.25	1.80	0.21	-0.03	-0.28
Zhwiki	4	2	.69	11.40	.83	.60	3.06	1.04	0.13	1.22	-0.31
Zhwiki	4	3	.55	8.69	.59	.53	2.34	.77	0.03	0.93	-0.26
Zhwiki	4	4	.56	4.71	.28	.33	1.63	.47	0.33	0.45	-0.28
Zhwiki	16	1	1.09	4.51	1.89	1.05	4.34	1.83	0.07	0.02	0.08
Zhwiki	16	2	.62	6.76	.91	.55	2.82	1.10	0.10	0.58	-0.28
Zhwiki	16	3	.49	5.63	.62	.38	1.75	.59	0.15	0.57	0.05
Zhwiki	16	4	.47	3.84	.25	.33	1.71	.60	0.21	0.31	-0.51
Zhwiki	32	1	1.08	6.28	2.08	1.04	4.70	1.92	0.05	0.23	0.24
Zhwiki	32	2	.61	4.94	.96	.54	2.79	1.14	0.10	0.32	-0.26
Zhwiki	32	3	.45	4.15	.63	.36	1.97	.81	0.12	0.32	-0.26
Zhwiki	32	4	.47	3.30	.37	.29	1.50	.62	0.27	0.27	-0.37
									μ	**0.15**	0.43	-0.21
									σ	0.09	0.35	0.23

Table D.60: `base64` PAPI comparison between `2020` and `CURR` on `SSE-4.2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.32	42.60	28.00	1.08	42.50	27.10	0.16	0.01	0.61
Arwiki	4	2	.87	70.70	14.40	.65	29.60	13.80	0.15	2.87	0.39
Arwiki	4	3	.74	59.50	9.56	.51	19.90	7.63	0.16	2.76	1.35
Arwiki	4	4	.71	48.90	7.24	.38	16.70	7.03	0.23	2.25	0.14
Arwiki	16	1	1.17	48.60	27.80	1.08	42.30	27.70	0.06	0.44	0.07
Arwiki	16	2	.71	93.00	13.90	.58	25.80	14.10	0.09	4.69	-0.16
Arwiki	16	3	.59	74.10	9.12	.38	14.40	7.13	0.15	4.17	1.38
Arwiki	16	4	.60	63.00	6.76	.34	13.60	7.26	0.18	3.45	-0.35
Arwiki	32	1	1.14	96.00	27.50	1.08	51.20	27.80	0.04	3.13	-0.23
Arwiki	32	2	.66	84.20	13.80	.56	28.30	14.10	0.07	3.90	-0.22
Arwiki	32	3	.52	69.00	9.19	.37	17.00	9.00	0.11	3.63	0.13
Arwiki	32	4	.54	58.70	6.89	.30	14.60	7.15	0.17	3.08	-0.18
Enwiki	4	1	.91	29.70	19.30	.76	29.30	18.70	0.15	0.03	0.66
Enwiki	4	2	.60	50.70	9.91	.45	20.50	9.52	0.15	3.04	0.40
Enwiki	4	3	.53	40.10	6.68	.35	13.80	5.28	0.17	2.65	1.41
Enwiki	4	4	.50	33.20	5.05	.27	11.80	4.86	0.24	2.16	0.19
Enwiki	16	1	.81	35.30	19.20	.76	29.00	18.90	0.05	0.63	0.31
Enwiki	16	2	.48	64.30	9.67	.40	17.80	9.76	0.08	4.69	-0.09
Enwiki	16	3	.42	51.00	6.35	.26	9.93	4.93	0.15	4.13	1.43
Enwiki	16	4	.41	41.80	4.75	.24	9.39	5.02	0.17	3.27	-0.27
Enwiki	32	1	.79	65.40	19.00	.76	34.90	19.00	0.04	3.07	0.01
Enwiki	32	2	.45	58.90	9.55	.39	19.60	9.82	0.06	3.96	-0.28
Enwiki	32	3	.37	48.10	6.38	.26	11.90	6.23	0.11	3.64	0.15
Enwiki	32	4	.38	41.20	4.80	.22	10.60	4.96	0.17	3.08	-0.16
Ruwiki	4	1	1.62	52.60	34.40	1.34	52.40	33.40	0.16	0.02	0.57
Ruwiki	4	2	1.06	86.70	17.70	.80	36.50	17.10	0.15	2.85	0.37
Ruwiki	4	3	.91	73.70	11.80	.63	24.50	9.36	0.16	2.79	1.40
Ruwiki	4	4	.88	61.10	8.93	.47	20.70	8.70	0.23	2.29	0.13
Ruwiki	16	1	1.44	63.80	34.30	1.33	52.20	34.00	0.06	0.66	0.14
Ruwiki	16	2	.88	112.00	17.20	.71	31.80	17.40	0.09	4.52	-0.13
Ruwiki	16	3	.76	87.10	11.40	.47	17.70	8.78	0.16	3.93	1.47
Ruwiki	16	4	.74	73.20	8.45	.42	16.70	8.90	0.18	3.20	-0.26
Ruwiki	32	1	1.40	102.00	34.00	1.33	59.60	34.20	0.04	2.43	-0.12
Ruwiki	32	2	.83	106.00	17.00	.69	36.20	17.40	0.08	3.96	-0.20
Ruwiki	32	3	.67	87.80	11.40	.45	20.90	11.10	0.12	3.79	0.17
Ruwiki	32	4	.67	71.70	8.46	.37	18.40	8.81	0.17	3.02	-0.20
Zhwiki	4	1	.62	20.20	13.30	.52	20.20	12.90	0.16	0.01	0.64
Zhwiki	4	2	.41	33.50	6.86	.31	14.10	6.59	0.14	2.85	0.39
Zhwiki	4	3	.36	26.90	4.66	.24	9.47	3.61	0.18	2.55	1.54
Zhwiki	4	4	.34	22.90	3.48	.19	8.02	3.38	0.22	2.18	0.15
Zhwiki	16	1	.56	23.90	13.20	.52	20.10	13.10	0.06	0.56	0.09
Zhwiki	16	2	.34	42.00	6.69	.28	12.30	6.72	0.10	4.36	-0.05
Zhwiki	16	3	.29	33.10	4.42	.18	6.81	3.38	0.17	3.86	1.53
Zhwiki	16	4	.29	27.90	3.30	.17	6.56	3.42	0.18	3.13	-0.18
Zhwiki	32	1	.54	40.50	13.10	.51	24.10	13.20	0.04	2.41	-0.21
Zhwiki	32	2	.34	41.80	6.65	.27	13.60	6.70	0.10	4.15	-0.07
Zhwiki	32	3	.26	33.80	4.42	.18	7.96	4.28	0.13	3.79	0.21
Zhwiki	32	4	.26	28.50	3.31	.15	7.06	3.40	0.16	3.15	-0.13
								μ	**0.13**	2.82	0.29
								σ	0.05	1.31	0.57

Table D.61: `base64` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	.98	22.70	22.40	.78	22.60	22.40	0.14	0.00	0.00
Arwiki	4	2	1.03	57.00	11.20	.55	14.70	11.20	0.33	2.96	0.01
Arwiki	4	3	.97	45.60	7.43	.43	9.47	6.60	0.38	2.52	0.58
Arwiki	4	4	.95	36.50	5.62	.43	8.01	5.61	0.36	1.99	0.01
Arwiki	16	1	.85	22.60	22.40	.78	22.60	22.40	0.05	0.00	0.00
Arwiki	16	2	.75	44.80	11.20	.48	13.00	11.20	0.19	2.22	0.01
Arwiki	16	3	.72	35.90	7.50	.35	6.92	5.65	0.26	2.02	1.30
Arwiki	16	4	.75	27.30	5.65	.34	6.84	5.59	0.28	1.43	0.04
Arwiki	32	1	.82	22.60	22.40	.77	22.60	22.40	0.03	0.00	0.00
Arwiki	32	2	.64	35.40	11.20	.45	12.20	11.20	0.13	1.62	-0.00
Arwiki	32	3	.60	31.50	7.47	.32	7.98	6.99	0.20	1.64	0.34
Arwiki	32	4	.68	27.70	5.66	.26	6.20	5.60	0.29	1.50	0.04
Enwiki	4	1	.68	15.70	15.50	.54	15.70	15.50	0.14	0.00	0.00
Enwiki	4	2	.70	39.50	7.76	.38	10.20	7.76	0.32	2.96	0.00
Enwiki	4	3	.64	32.50	5.16	.30	6.54	4.58	0.34	2.62	0.58
Enwiki	4	4	.64	25.30	3.90	.30	5.53	3.90	0.35	1.99	0.01
Enwiki	16	1	.59	15.70	15.50	.54	15.70	15.50	0.05	0.00	0.00
Enwiki	16	2	.51	31.00	7.76	.33	9.01	7.77	0.17	2.22	-0.02
Enwiki	16	3	.48	25.00	5.19	.24	4.79	3.91	0.25	2.03	1.29
Enwiki	16	4	.52	18.90	3.93	.24	4.76	3.88	0.28	1.42	0.04
Enwiki	32	1	.57	15.70	15.50	.54	15.70	15.50	0.03	0.00	0.00
Enwiki	32	2	.43	24.50	7.76	.30	8.45	7.73	0.13	1.62	0.03
Enwiki	32	3	.40	21.80	5.18	.22	5.53	4.84	0.19	1.64	0.34
Enwiki	32	4	.46	19.20	3.93	.17	4.30	3.88	0.29	1.51	0.05
Ruwiki	4	1	1.21	27.90	27.60	.96	27.90	27.60	0.14	0.00	0.00
Ruwiki	4	2	1.26	70.20	13.80	.69	18.10	13.90	0.33	2.95	-0.03
Ruwiki	4	3	1.17	57.30	9.19	.53	11.70	8.12	0.36	2.58	0.60
Ruwiki	4	4	1.16	45.00	6.92	.53	9.88	6.86	0.36	1.99	0.04
Ruwiki	16	1	1.05	27.90	27.60	.96	27.90	27.60	0.05	0.00	0.00
Ruwiki	16	2	.91	55.20	13.80	.59	16.10	13.70	0.18	2.22	0.03
Ruwiki	16	3	.87	44.20	9.22	.42	8.50	6.96	0.25	2.02	1.28
Ruwiki	16	4	.92	33.60	6.94	.42	8.47	6.91	0.28	1.42	0.02
Ruwiki	32	1	1.01	27.90	27.60	.95	27.90	27.60	0.03	0.00	0.00
Ruwiki	32	2	.79	43.70	13.80	.55	15.00	13.80	0.14	1.62	0.01
Ruwiki	32	3	.73	38.80	9.24	.39	9.84	8.61	0.19	1.64	0.35
Ruwiki	32	4	.83	34.00	6.95	.32	7.64	6.90	0.29	1.49	0.03
Zhwiki	4	1	.47	10.80	10.60	.37	10.80	10.60	0.14	0.00	0.00
Zhwiki	4	2	.49	27.10	5.33	.27	6.96	5.35	0.33	2.95	-0.04
Zhwiki	4	3	.45	22.00	3.54	.20	4.46	3.14	0.36	2.57	0.58
Zhwiki	4	4	.45	17.30	2.69	.21	3.77	2.66	0.35	1.99	0.04
Zhwiki	16	1	.41	10.80	10.60	.37	10.80	10.60	0.05	0.00	-0.00
Zhwiki	16	2	.36	21.30	5.33	.23	6.18	5.33	0.19	2.22	-0.01
Zhwiki	16	3	.33	17.10	3.57	.17	3.28	2.69	0.24	2.03	1.29
Zhwiki	16	4	.36	13.00	2.71	.16	3.25	2.67	0.28	1.43	0.07
Zhwiki	32	1	.39	10.80	10.60	.37	10.80	10.60	0.03	0.00	0.00
Zhwiki	32	2	.30	16.80	5.33	.21	5.80	5.33	0.13	1.62	0.00
Zhwiki	32	3	.28	15.00	3.56	.15	3.79	3.33	0.19	1.64	0.35
Zhwiki	32	4	.32	13.30	2.71	.12	2.95	2.66	0.28	1.51	0.07
								$\boldsymbol{\mu}$	**0.22**	1.50	0.20
								$\boldsymbol{\sigma}$	0.11	0.96	0.37

Table D.62: `base64` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.2 Case study: `csv2json`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Census	4	1				77.70	170.00	3.17			
Census	4	2	42.40	120.00	1.88	40.50	97.70	1.59	0.66	0.77	0.10
Census	4	3	29.10	83.10	1.36	27.80	67.80	.97	0.46	0.54	0.14
Census	4	4	26.20	64.50	1.19	20.30	50.90	.73	2.06	0.48	0.16
Census	16	1	77.90	96.60	5.00	74.70	86.90	3.39	1.14	0.34	0.57
Census	16	2	39.00	55.20	4.37	37.80	49.20	1.73	0.45	0.21	0.93
Census	16	3	27.30	42.00	4.48	24.80	31.80	1.31	0.88	0.36	1.11
Census	16	4	23.60	34.30	3.77	18.60	25.20	1.45	1.75	0.32	0.81
Census	32	1	76.60	95.50	9.73	74.90	74.50	3.34	0.62	0.73	2.24
Census	32	2	37.70	56.30	6.65	36.80	43.60	2.69	0.33	0.45	1.39
Census	32	3	27.90	41.00	5.65	24.60	28.40	3.56	1.16	0.44	0.73
Census	32	4	23.80	33.50	4.93	18.50	21.60	3.57	1.87	0.42	0.48
Fin	4	1	.54	1.62	.02	.50	1.22	.02	3.15	3.06	0.09
Fin	4	2	.31	.92	.02	.27	.71	.01	2.74	1.56	0.76
Fin	4	3	.21	.67	.03	.18	.48	.01	1.97	1.42	1.36
Fin	4	4	.19	.52	.02	.15	.39	.01	3.53	0.99	1.34
Fin	16	1	.49	.72	.06	.48	.58	.02	1.14	1.07	2.47
Fin	16	2	.26	.41	.04	.24	.31	.02	1.19	0.74	1.83
Fin	16	3	.20	.30	.03	.17	.22	.02	2.56	0.64	1.06
Fin	16	4	.18	.24	.02	.13	.17	.02	4.05	0.56	0.53
Fin	32	1	.49	.61	.07	.48	.48	.03	0.92	1.01	3.44
Fin	32	2	.27	.35	.04	.24	.25	.03	1.98	0.71	1.14
Fin	32	3	.21	.25	.03	.17	.18	.03	3.40	0.55	0.32
Fin	32	4	.18	.21	.03	.13	.14	.03	4.18	0.52	0.29
Trade	4	1	8.50	23.10	.38	7.97	18.30	.37	2.28	2.08	0.06
Trade	4	2	4.68	13.50	.28	4.26	10.40	.17	1.80	1.32	0.44
Trade	4	3	3.28	9.37	.35	2.82	6.84	.12	1.98	1.08	1.00
Trade	4	4	2.80	7.36	.34	2.19	5.39	.09	2.63	0.85	1.09
Trade	16	1	8.04	10.10	.84	7.66	8.51	.33	1.60	0.70	2.19
Trade	16	2	4.21	6.17	.63	3.84	4.59	.18	1.59	0.68	1.94
Trade	16	3	3.23	4.53	.49	2.56	3.10	.19	2.89	0.61	1.29
Trade	16	4	2.59	3.61	.42	1.94	2.46	.23	2.79	0.49	0.85
Trade	32	1	7.97	9.47	1.14	7.56	7.16	.36	1.76	0.99	3.32
Trade	32	2	4.07	5.58	.75	3.80	4.02	.42	1.16	0.67	1.43
Trade	32	3	3.12	3.98	.58	2.54	2.61	.42	2.49	0.58	0.72
Trade	32	4	2.75	3.11	.46	1.92	2.07	.37	3.57	0.45	0.39
								μ	**1.96**	0.81	1.09
								σ	1.04	0.55	0.85

Table D.63: `csv2json` PAPI comparison between `2020` and `CURR` on `SSE-4.2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Census	4	1	25.20	2070.00	17.90	20.30	1180.00	17.50	1.69	31.40	0.14
Census	4	2	13.60	1150.00	9.16	11.80	725.00	8.31	0.61	15.00	0.30
Census	4	3	8.73	771.00	14.70	7.67	471.00	5.45	0.37	10.50	3.25
Census	4	4	6.99	596.00	29.50	6.21	365.00	4.10	0.27	8.08	8.93
Census	16	1	21.00	1540.00	22.00	19.60	1050.00	18.30	0.48	17.00	1.28
Census	16	2	10.50	784.00	117.00	9.89	542.00	9.07	0.20	8.51	37.80
Census	16	3	7.07	533.00	142.00	6.71	352.00	7.00	0.12	6.35	47.40
Census	16	4	5.35	398.00	132.00	5.06	266.00	11.70	0.10	4.65	42.20
Census	32	1	20.30	1550.00	199.00	19.40	1190.00	18.70	0.29	12.90	63.40
Census	32	2	10.20	792.00	272.00	9.79	600.00	19.40	0.16	6.71	88.50
Census	32	3	6.95	528.00	225.00	6.58	404.00	53.30	0.13	4.35	60.30
Census	32	4	5.28	386.00	183.00	4.99	321.00	69.70	0.10	2.28	39.80
Fin	4	1	.17	14.20	.18	.14	8.44	.12	2.72	44.40	4.53
Fin	4	2	.11	8.03	.69	.09	5.09	.06	1.62	22.50	48.40
Fin	4	3	.08	5.55	.77	.06	3.40	.04	1.47	16.50	55.90
Fin	4	4	.08	4.30	.73	.06	2.75	.04	0.94	11.90	53.50
Fin	16	1	.15	10.00	1.76	.13	6.75	.14	1.06	25.20	124.00
Fin	16	2	.09	5.29	1.51	.07	3.53	.13	1.28	13.40	106.00
Fin	16	3	.08	3.69	1.21	.05	2.36	.26	1.92	10.20	73.00
Fin	16	4	.07	2.87	1.02	.05	1.89	.37	1.10	7.49	49.90
Fin	32	1	.15	9.42	2.86	.14	6.61	.24	0.96	21.50	201.00
Fin	32	2	.09	4.91	1.86	.07	3.47	.69	1.61	11.00	89.90
Fin	32	3	.09	3.40	1.41	.05	2.30	.72	2.38	8.44	53.50
Fin	32	4	.08	2.71	1.21	.06	1.83	.65	2.09	6.72	43.00
Trade	4	1	2.61	218.00	1.91	2.12	128.00	1.68	2.07	38.90	0.99
Trade	4	2	1.46	122.00	8.64	1.28	76.70	.81	0.76	19.20	33.60
Trade	4	3	1.11	83.50	11.40	.95	52.30	.54	0.68	13.40	46.50
Trade	4	4	.89	62.50	11.00	.74	39.60	.40	0.64	9.81	45.30
Trade	16	1	2.19	157.00	25.00	2.04	104.00	1.78	0.63	22.60	99.70
Trade	16	2	1.14	79.80	24.00	1.04	53.60	1.06	0.43	11.30	98.50
Trade	16	3	.80	53.50	19.30	.71	36.00	2.38	0.41	7.52	72.40
Trade	16	4	.66	40.60	16.10	.56	27.70	4.34	0.42	5.56	50.40
Trade	32	1	2.14	154.00	49.40	2.03	112.00	2.09	0.49	18.10	203.00
Trade	32	2	1.14	77.20	32.10	1.03	57.30	7.86	0.45	8.54	104.00
Trade	32	3	.85	52.30	23.80	.70	38.50	10.20	0.63	5.88	58.10
Trade	32	4	.72	39.70	18.90	.55	30.10	9.54	0.72	4.10	40.20
								$\boldsymbol{\mu}$	**0.89**	13.70	59.70
								$\boldsymbol{\sigma}$	0.70	9.47	47.20

Table D.64: `csv2json` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Census	4	1	16.10	422.00	13.50	12.00	33.80	13.40	1.45	13.60	0.04
Census	4	2	11.10	418.00	14.30	9.80	234.00	8.12	0.45	6.46	2.16
Census	4	3	7.47	296.00	20.40	7.01	174.00	6.02	0.16	4.25	5.04
Census	4	4	6.67	238.00	28.10	6.45	149.00	4.77	0.08	3.13	8.18
Census	16	1	13.70	554.00	27.90	11.90	253.00	13.40	0.65	10.60	5.07
Census	16	2	7.16	315.00	78.60	6.34	165.00	7.84	0.29	5.23	24.80
Census	16	3	5.98	209.00	83.00	4.38	114.00	5.73	0.56	3.35	27.10
Census	16	4	6.69	157.00	73.60	3.46	88.60	5.23	1.13	2.41	24.00
Census	32	1	13.70	614.00	123.00	12.50	330.00	13.60	0.42	9.96	38.50
Census	32	2	7.66	311.00	140.00	6.15	174.00	10.90	0.53	4.79	45.30
Census	32	3	7.57	208.00	111.00	4.29	118.00	15.70	1.15	3.14	33.40
Census	32	4	8.09	156.00	89.40	3.29	89.30	19.40	1.68	2.33	24.60
Fin	4	1	.12	2.59	.27	.09	.94	.09	2.34	12.70	13.10
Fin	4	2	.10	2.51	.56	.08	1.22	.06	1.45	9.92	37.90
Fin	4	3	.07	1.89	.49	.07	1.01	.05	0.10	6.75	33.70
Fin	4	4	.07	1.62	.47	.07	.79	.05	-0.12	6.34	32.00
Fin	16	1	.10	3.42	1.06	.09	1.79	.10	1.29	12.50	73.40
Fin	16	2	.07	2.16	1.02	.05	1.20	.10	1.55	7.35	70.40
Fin	16	3	.07	1.47	.77	.04	.79	.12	2.20	5.24	49.90
Fin	16	4	.07	1.13	.61	.03	.63	.15	2.91	3.81	35.30
Fin	32	1	.10	3.89	1.71	.09	2.22	.17	1.18	12.80	118.00
Fin	32	2	.08	2.16	1.19	.05	1.19	.29	2.11	7.45	68.70
Fin	32	3	.08	1.43	.84	.04	.77	.31	3.01	5.04	41.00
Fin	32	4	.12	1.12	.69	.04	.59	.28	5.99	4.04	31.40
Trade	4	1	1.71	45.00	2.66	1.29	11.70	1.29	1.79	14.30	5.87
Trade	4	2	1.23	39.50	7.57	.99	20.70	.86	1.00	8.05	28.80
Trade	4	3	.99	30.10	7.80	.86	16.30	.69	0.55	5.93	30.50
Trade	4	4	.82	23.10	6.98	.80	13.20	.55	0.11	4.26	27.60
Trade	16	1	1.49	58.10	16.20	1.28	28.30	1.30	0.89	12.80	63.90
Trade	16	2	.91	32.50	15.00	.72	17.30	.89	0.81	6.52	60.50
Trade	16	3	.80	21.40	11.30	.48	11.70	1.02	1.37	4.16	44.20
Trade	16	4	.84	16.20	9.00	.39	8.95	1.29	1.96	3.12	33.00
Trade	32	1	1.47	61.90	26.00	1.34	35.00	1.53	0.53	11.50	105.00
Trade	32	2	.90	31.30	17.30	.67	18.00	2.89	1.01	5.72	61.80
Trade	32	3	.88	20.90	12.50	.47	12.00	3.72	1.77	3.82	37.80
Trade	32	4	.92	15.60	9.71	.40	9.11	3.65	2.24	2.80	26.00
								μ	**1.29**	6.84	38.00
								σ	1.13	3.60	26.20

Table D.65: `csv2json` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.3 Case study: `editd`

CONFIG			2020			CURR			NORM. DIFFERENCE		
DIST	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
1	4	1	20.20	276.00	1.21	19.20	285.00	1.21	4.20	-3.48	0.00
1	4	2	11.10	145.00	.59	9.70	145.00	.60	5.52	0.09	-0.06
1	4	3	7.89	99.30	.38	6.43	96.70	.40	5.72	1.03	-0.08
1	4	4	6.71	76.30	.25	4.92	74.30	.30	7.04	0.78	-0.18
1	16	1	17.20	216.00	1.02	17.00	217.00	1.01	0.78	-0.46	0.03
1	16	2	9.26	119.00	.61	8.59	111.00	.51	2.62	3.36	0.39
1	16	3	7.02	74.20	.40	5.77	73.40	.34	4.92	0.32	0.22
1	16	4	5.28	60.90	.24	4.31	55.60	.26	3.81	2.09	-0.09
1	32	1	16.70	228.00	.96	16.70	214.00	.97	-0.25	5.60	-0.04
1	32	2	9.61	117.00	.67	8.39	108.00	.50	4.79	3.28	0.67
1	32	3	6.63	75.30	.37	5.56	71.00	.33	4.21	1.67	0.17
1	32	4	5.48	57.20	.25	4.20	53.40	.31	5.05	1.49	-0.24
2	4	1	33.50	456.00	1.21	33.00	456.00	1.18	1.77	0.13	0.12
2	4	2	17.60	236.00	.60	16.30	230.00	.59	5.12	2.62	0.06
2	4	3	12.80	162.00	.38	11.10	153.00	.40	6.90	3.51	-0.06
2	4	4	10.70	123.00	.26	8.48	118.00	.29	8.74	2.13	-0.12
2	16	1	29.60	398.00	1.02	30.00	396.00	1.01	-1.55	0.73	0.05
2	16	2	15.70	196.00	.53	15.20	202.00	.52	2.32	-2.26	0.04
2	16	3	12.00	133.00	.35	10.10	131.00	.34	7.66	0.61	0.02
2	16	4	9.94	105.00	.23	7.50	99.80	.30	9.57	2.12	-0.29
2	32	1	29.00	374.00	.95	29.70	384.00	1.00	-2.94	-3.70	-0.21
2	32	2	15.40	195.00	.48	15.10	194.00	.49	1.44	0.29	-0.04
2	32	3	11.80	132.00	.48	9.81	127.00	.45	7.84	2.05	0.13
2	32	4	9.62	101.00	1.05	7.32	95.00	1.11	9.03	2.51	-0.21
4	4	1	71.80	613.00	1.24	70.90	597.00	1.24	3.59	6.52	-0.00
4	4	2	37.20	312.00	.62	36.50	304.00	.62	2.77	3.31	-0.00
4	4	3	28.50	211.00	.38	25.40	204.00	.41	12.10	2.88	-0.11
4	4	4	22.10	159.00	.27	18.30	154.00	.31	15.20	2.30	-0.14
4	16	1	65.70	504.00	1.05	66.00	517.00	1.04	-1.38	-5.04	0.02
4	16	2	34.30	262.00	.60	34.30	261.00	.58	-0.13	0.13	0.09
4	16	3	25.70	176.00	.75	22.00	174.00	.73	14.50	0.74	0.10
4	16	4	21.40	133.00	1.44	16.50	130.00	1.88	19.40	1.29	-1.74
4	32	1	64.50	498.00	1.06	64.70	501.00	1.07	-0.80	-1.03	-0.04
4	32	2	35.60	251.00	3.30	32.80	249.00	1.85	11.40	0.63	5.73
4	32	3	28.00	182.00	17.20	22.20	163.00	12.70	22.90	7.67	17.60
4	32	4	22.30	127.00	28.00	17.40	121.00	23.40	19.10	2.56	18.00
8	4	1	180.00	942.00	17.80	178.00	915.00	17.90	6.18	10.50	-0.38
8	4	2	95.60	501.00	13.30	96.20	479.00	12.90	-2.59	8.66	1.68
8	4	3	184.00	333.00	8.96	69.30	326.00	9.67	452.00	2.55	-2.79
8	4	4	266.00	255.00	4.35	58.70	248.00	12.30	815.00	2.72	-31.20
8	16	1	173.00	842.00	19.80	172.00	813.00	18.90	3.93	11.60	3.65
8	16	2	213.00	416.00	15.30	177.00	398.00	14.10	142.00	6.80	4.77
8	16	3	144.00	286.00	39.40	66.70	290.00	43.60	303.00	-1.83	-16.70
8	16	4	241.00	207.00	49.20	143.00	203.00	51.20	385.00	1.70	-7.88
8	32	1	173.00	812.00	37.10	172.00	785.00	33.30	2.63	10.50	14.90
8	32	2	93.70	406.00	93.90	92.90	403.00	93.60	2.89	1.15	1.02
8	32	3	257.00	256.00	115.00	157.00	252.00	107.00	394.00	1.77	31.70
8	32	4	289.00	187.00	104.00	152.00	180.00	104.00	536.00	2.76	-0.14
								μ	**68.00**	2.24	0.80
								σ	168.00	3.44	8.20

Table D.66: `editd` PAPI comparison between `2020` and `CURR` on `SSE-4.2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
DIST	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
1	4	1	13.10	1290.00	5.40	12.00	1180.00	5.51	4.19	42.20	-0.44
1	4	2	6.65	675.00	2.80	6.08	605.00	2.86	2.24	27.80	-0.25
1	4	3	4.53	462.00	1.88	4.08	406.00	1.91	1.77	22.00	-0.09
1	4	4	3.49	343.00	1.41	3.09	306.00	1.44	1.55	14.90	-0.14
1	16	1	11.30	616.00	5.34	11.10	558.00	5.53	1.01	22.90	-0.77
1	16	2	5.70	313.00	2.68	5.56	283.00	2.77	0.56	11.70	-0.35
1	16	3	3.81	212.00	1.79	3.71	188.00	1.80	0.41	9.28	-0.07
1	16	4	2.93	160.00	1.33	2.80	143.00	1.37	0.52	6.76	-0.17
1	32	1	11.00	546.00	5.32	10.90	533.00	5.41	0.44	5.31	-0.35
1	32	2	5.64	275.00	2.69	5.48	267.00	2.72	0.60	3.20	-0.12
1	32	3	3.71	183.00	1.84	3.65	177.00	1.77	0.21	2.32	0.28
1	32	4	2.80	138.00	1.44	2.75	132.00	1.36	0.20	2.11	0.33
2	4	1	19.80	1820.00	5.32	18.80	1670.00	5.42	4.11	56.50	-0.40
2	4	2	10.10	945.00	2.78	9.46	845.00	2.80	2.33	39.30	-0.09
2	4	3	6.77	633.00	1.88	6.34	570.00	1.86	1.72	24.70	0.09
2	4	4	5.13	473.00	1.40	4.78	429.00	1.40	1.35	17.20	-0.01
2	16	1	17.90	1010.00	5.33	17.80	1010.00	5.57	0.62	-1.28	-0.94
2	16	2	9.02	515.00	2.68	8.92	512.00	2.78	0.41	1.37	-0.39
2	16	3	6.06	350.00	1.79	5.96	343.00	1.83	0.43	2.87	-0.13
2	16	4	4.54	253.00	1.34	4.49	259.00	1.41	0.22	-2.30	-0.30
2	32	1	17.60	867.00	5.31	17.50	819.00	5.40	0.64	19.20	-0.36
2	32	2	8.84	431.00	2.72	8.76	411.00	2.69	0.30	7.84	0.11
2	32	3	5.95	302.00	2.32	5.85	276.00	1.80	0.36	10.60	2.06
2	32	4	4.48	218.00	4.16	4.41	207.00	1.93	0.27	4.17	8.75
4	4	1	40.20	2670.00	5.44	39.40	2490.00	5.52	3.16	69.10	-0.31
4	4	2	20.30	1370.00	2.83	19.80	1270.00	2.86	1.72	41.30	-0.11
4	4	3	13.60	924.00	1.86	13.20	851.00	1.87	1.35	28.70	-0.01
4	4	4	10.20	709.00	1.39	9.96	638.00	1.42	1.11	27.90	-0.11
4	16	1	38.00	1650.00	5.53	37.90	1570.00	5.54	0.36	29.50	-0.04
4	16	2	19.10	846.00	2.89	19.00	790.00	2.83	0.51	21.90	0.23
4	16	3	12.80	551.00	2.65	12.70	529.00	2.38	0.32	8.50	1.09
4	16	4	9.62	405.00	4.68	9.55	398.00	4.81	0.26	2.49	-0.50
4	32	1	37.70	1460.00	5.58	37.60	1370.00	5.54	0.24	37.20	0.13
4	32	2	18.90	707.00	9.27	18.90	692.00	5.57	-0.04	5.90	14.60
4	32	3	12.90	508.00	52.90	12.70	462.00	23.60	0.49	18.00	115.00
4	32	4	9.74	354.00	84.00	9.86	343.00	72.00	-0.47	4.27	47.30
8	4	1	102.00	4910.00	169.00	105.00	4960.00	169.00	-10.50	-20.10	0.07
8	4	2	56.10	2600.00	116.00	57.20	2570.00	122.00	-4.31	14.00	-22.10
8	4	3	41.10	1730.00	71.90	41.30	1730.00	89.70	-0.80	-0.17	-69.90
8	4	4	33.70	1310.00	47.40	32.80	1350.00	109.00	3.38	-15.80	-244.00
8	16	1	100.00	3150.00	182.00	104.00	3030.00	171.00	-13.60	48.80	45.20
8	16	2	57.10	1540.00	109.00	58.70	1470.00	84.90	-6.23	29.40	96.60
8	16	3	40.30	1100.00	240.00	40.10	1130.00	267.00	1.11	-10.30	-107.00
8	16	4	33.60	786.00	238.00	34.70	765.00	260.00	-4.19	8.23	-87.90
8	32	1	101.00	2830.00	291.00	103.00	2680.00	201.00	-8.55	58.90	354.00
8	32	2	55.40	1430.00	463.00	57.10	1410.00	484.00	-6.86	5.74	-81.30
8	32	3	42.40	936.00	396.00	44.00	865.00	425.00	-6.52	28.10	-114.00
8	32	4	34.90	663.00	349.00	36.40	612.00	330.00	-5.94	20.00	75.40
								μ	**-0.57**	16.90	0.61
								σ	3.66	18.70	73.80

Table D.67: `editd` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
DIST	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
1	4	1	7.96	73.60	4.00	7.01	55.90	4.00	3.74	6.97	0.01
1	4	2	4.76	69.40	2.22	4.11	38.30	2.11	2.57	12.20	0.42
1	4	3	3.25	53.60	1.48	2.81	26.10	1.41	1.75	10.80	0.29
1	4	4	2.62	57.30	1.09	2.13	21.40	1.05	1.92	14.10	0.15
1	16	1	6.35	41.70	4.00	6.09	31.50	4.00	1.00	4.00	-0.00
1	16	2	3.31	26.60	2.11	3.17	19.50	2.10	0.54	2.78	0.05
1	16	3	2.32	57.50	1.37	2.13	14.10	1.40	0.74	17.10	-0.10
1	16	4	1.70	14.70	1.06	1.61	10.80	1.05	0.35	1.52	0.05
1	32	1	6.24	113.00	4.00	6.07	88.00	4.01	0.66	10.00	-0.03
1	32	2	3.15	60.30	2.03	3.06	44.70	2.02	0.33	6.14	0.03
1	32	3	2.11	40.30	1.36	2.06	32.20	1.36	0.22	3.17	-0.02
1	32	4	1.61	30.70	1.03	1.57	24.40	1.03	0.15	2.45	-0.02
2	4	1	12.10	204.00	4.03	11.10	157.00	4.01	4.01	18.50	0.08
2	4	2	7.04	172.00	2.34	6.28	96.00	2.27	2.96	29.80	0.31
2	4	3	4.98	133.00	1.63	4.21	64.50	1.47	3.04	26.90	0.62
2	4	4	3.76	78.20	1.14	3.20	49.30	1.09	2.21	11.40	0.21
2	16	1	10.40	215.00	4.01	10.10	198.00	4.01	1.02	6.76	-0.02
2	16	2	5.48	170.00	2.50	5.13	104.00	2.04	1.37	25.70	1.81
2	16	3	3.58	76.00	2.00	3.44	71.30	1.38	0.55	1.86	2.45
2	16	4	2.72	58.80	1.33	2.60	51.90	1.04	0.44	2.73	1.15
2	32	1	10.50	320.00	4.01	10.10	264.00	4.01	1.61	21.90	-0.01
2	32	2	5.18	155.00	2.03	5.05	132.00	2.02	0.53	8.69	0.03
2	32	3	3.50	102.00	1.37	3.38	89.30	1.36	0.44	4.82	0.01
2	32	4	2.64	79.10	1.07	2.56	67.20	1.05	0.33	4.67	0.10
4	4	1	23.80	478.00	4.15	22.80	405.00	4.12	4.11	28.70	0.09
4	4	2	12.80	378.00	3.19	12.10	237.00	2.50	2.77	55.50	2.71
4	4	3	8.81	262.00	2.71	8.15	162.00	1.71	2.59	39.50	3.94
4	4	4	6.73	205.00	1.84	6.15	122.00	1.30	2.27	32.30	2.12
4	16	1	22.60	631.00	4.13	21.90	532.00	4.13	2.65	38.80	0.00
4	16	2	11.20	292.00	2.39	11.00	271.00	2.12	0.74	8.44	1.07
4	16	3	7.56	187.00	1.76	7.39	182.00	1.52	0.64	1.77	0.94
4	16	4	5.71	148.00	1.16	5.58	137.00	1.23	0.53	4.31	-0.27
4	32	1	22.50	599.00	4.21	22.20	564.00	4.19	1.21	13.90	0.09
4	32	2	11.20	298.00	2.78	11.10	285.00	2.67	0.33	5.20	0.45
4	32	3	7.69	199.00	10.00	7.48	191.00	3.97	0.83	2.95	23.80
4	32	4	5.78	151.00	9.48	5.74	144.00	8.07	0.15	2.93	5.55
8	4	1	60.40	1580.00	139.00	61.50	1670.00	139.00	-4.26	-35.30	-0.33
8	4	2	37.20	946.00	104.00	37.20	889.00	101.00	0.14	22.30	13.40
8	4	3	34.60	583.00	55.00	30.50	607.00	73.00	15.90	-9.37	-70.90
8	4	4	33.50	476.00	20.60	25.20	492.00	86.50	32.50	-6.27	-259.00
8	16	1	58.50	1360.00	143.00	59.40	1320.00	142.00	-3.78	15.30	2.41
8	16	2	42.50	653.00	46.60	41.80	632.00	46.40	2.99	8.42	0.66
8	16	3	36.20	400.00	17.60	27.20	500.00	121.00	35.30	-39.50	-405.00
8	16	4	33.20	324.00	39.20	30.50	335.00	63.20	10.60	-4.34	-94.60
8	32	1	58.30	1210.00	162.00	59.50	1190.00	160.00	-4.62	6.15	8.03
8	32	2	35.50	644.00	218.00	36.10	639.00	207.00	-2.32	2.01	40.90
8	32	3	31.10	427.00	205.00	35.00	390.00	158.00	-15.10	14.70	184.00
8	32	4	29.30	339.00	202.00	33.80	271.00	133.00	-18.00	26.50	274.00
								μ	**2.10**	10.20	-5.41
								σ	8.16	16.10	86.10

Table D.68: `editd` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.4 Case study: gb18030

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1				84.20	366.00	3.91			
Arwiki	4	2	50.10	342.00	2.64	43.20	217.00	1.83	3.11	5.61	0.36
Arwiki	4	3	34.70	238.00	1.59	28.90	161.00	1.23	2.63	3.50	0.16
Arwiki	4	4	28.90	190.00	1.00	21.60	122.00	.92	3.33	3.04	0.04
Arwiki	16	1				79.10	282.00	8.89			
Arwiki	16	2	45.60	243.00	4.53	39.50	158.00	4.85	2.73	3.85	-0.15
Arwiki	16	3	30.60	166.00	4.59	26.30	97.80	4.54	1.92	3.07	0.02
Arwiki	16	4	26.60	134.00	6.10	20.10	80.50	4.62	2.95	2.41	0.67
Arwiki	32	1				77.20	249.00	9.49			
Arwiki	32	2	42.10	235.00	11.50	39.10	137.00	8.52	1.36	4.44	1.35
Arwiki	32	3	30.00	159.00	16.90	26.10	87.40	7.96	1.74	3.22	4.02
Arwiki	32	4	27.60	141.00	24.30	20.10	68.50	7.68	3.40	3.26	7.50
Enwiki	4	1				47.90	255.00	1.96			
Enwiki	4	2	28.40	229.00	1.02	25.30	162.00	1.02	3.12	6.67	0.00
Enwiki	4	3	20.70	159.00	.65	16.00	94.10	.65	4.70	6.56	0.00
Enwiki	4	4	17.20	129.00	.47	12.40	81.40	.53	4.76	4.81	-0.06
Enwiki	16	1	48.80	421.00	3.94	46.10	217.00	3.98	2.73	20.60	-0.04
Enwiki	16	2	25.10	222.00	2.17	23.10	125.00	2.31	1.98	9.75	-0.14
Enwiki	16	3	17.20	149.00	2.49	15.40	76.80	2.37	1.80	7.27	0.11
Enwiki	16	4	14.60	118.00	4.32	12.10	62.50	2.94	2.55	5.60	1.39
Enwiki	32	1	47.20	466.00	4.22	45.40	269.00	4.66	1.84	19.90	-0.44
Enwiki	32	2	24.90	239.00	8.75	23.10	141.00	5.98	1.79	9.96	2.78
Enwiki	32	3	18.60	159.00	17.00	15.50	84.90	6.66	3.04	7.48	10.40
Enwiki	32	4	19.00	152.00	30.30	12.20	68.30	6.91	6.85	8.44	23.50
Ruwiki	4	1	96.20	587.00	3.65	82.50	396.00	3.58	7.75	10.80	0.04
Ruwiki	4	2	48.60	338.00	1.89	43.80	261.00	1.79	2.72	4.33	0.06
Ruwiki	4	3	33.90	231.00	1.19	27.80	150.00	1.16	3.46	4.55	0.01
Ruwiki	4	4	29.00	178.00	.84	21.50	131.00	.93	4.23	2.68	-0.05
Ruwiki	16	1				77.20	289.00	7.12			
Ruwiki	16	2	43.00	266.00	3.80	38.90	167.00	4.00	2.32	5.57	-0.12
Ruwiki	16	3	30.30	181.00	4.54	25.90	101.00	4.03	2.45	4.52	0.28
Ruwiki	16	4	26.20	143.00	6.67	19.70	84.30	4.85	3.62	3.33	1.03
Ruwiki	32	1				76.10	282.00	7.98			
Ruwiki	32	2	42.60	260.00	13.10	38.50	152.00	8.69	2.32	6.06	2.47
Ruwiki	32	3	30.30	175.00	20.30	25.90	97.00	9.27	2.44	4.42	6.21
Ruwiki	32	4	28.30	160.00	32.00	19.80	78.10	10.10	4.77	4.61	12.30
Zhwiki	4	1	63.30	342.00	1.42	58.90	303.00	1.39	7.87	6.92	0.04
Zhwiki	4	2	32.90	184.00	.71	30.40	161.00	.70	4.48	4.24	0.01
Zhwiki	4	3	23.90	133.00	.46	20.60	110.00	.47	5.85	3.97	-0.02
Zhwiki	4	4	19.20	106.00	.34	15.80	84.50	.35	6.11	3.83	-0.02
Zhwiki	16	1	55.10	224.00	2.24	53.90	173.00	2.26	2.14	9.15	-0.05
Zhwiki	16	2	28.50	115.00	1.46	27.30	92.70	1.40	2.12	3.95	0.10
Zhwiki	16	3	19.10	75.90	2.03	18.50	60.80	1.62	1.16	2.70	0.73
Zhwiki	16	4	17.80	71.00	3.05	14.10	47.20	1.95	6.58	4.25	1.96
Zhwiki	32	1	53.50	218.00	3.19	52.80	177.00	2.80	1.30	7.46	0.71
Zhwiki	32	2	28.20	111.00	5.73	26.80	90.30	3.52	2.38	3.73	3.97
Zhwiki	32	3	19.80	77.60	7.37	18.20	60.60	3.84	2.74	3.05	6.32
Zhwiki	32	4	17.70	73.00	7.84	14.40	48.10	4.01	6.01	4.45	6.86
								μ	**3.41**	5.90	2.25
								σ	1.75	3.83	4.47

Table D.69: gb18030 PAPI comparison between 2020 and CURR on SSE-4.2

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1				26.20	1940.00	46.30			
Arwiki	4	2	19.90	2490.00	22.70	14.10	1260.00	21.80	2.61	55.40	0.41
Arwiki	4	3	13.30	1720.00	14.80	10.30	931.00	14.40	1.36	35.50	0.15
Arwiki	4	4	10.20	1340.00	11.00	7.61	700.00	10.80	1.17	28.80	0.08
Arwiki	16	1				26.20	1730.00	46.60			
Arwiki	16	2	15.20	1680.00	25.30	13.50	942.00	25.00	0.74	33.40	0.17
Arwiki	16	3	10.10	1140.00	45.20	9.05	630.00	31.80	0.49	23.20	6.04
Arwiki	16	4	7.91	875.00	73.20	6.97	488.00	43.80	0.42	17.50	13.30
Arwiki	32	1				26.40	1650.00	48.80			
Arwiki	32	2	14.70	1590.00	150.00	13.50	862.00	81.10	0.51	32.90	31.10
Arwiki	32	3	10.00	1070.00	236.00	9.11	587.00	97.10	0.40	22.00	62.90
Arwiki	32	4	7.86	803.00	249.00	6.95	441.00	104.00	0.41	16.30	65.80
Enwiki	4	1	22.60	2990.00	20.50	14.80	1370.00	20.50	7.88	163.00	0.02
Enwiki	4	2	11.00	1600.00	10.10	8.45	855.00	9.91	2.57	74.90	0.18
Enwiki	4	3	7.25	1090.00	6.60	5.30	541.00	6.38	1.96	55.50	0.22
Enwiki	4	4	5.65	838.00	4.94	4.37	440.00	4.91	1.29	40.00	0.04
Enwiki	16	1				15.80	1190.00	20.90			
Enwiki	16	2	8.98	1190.00	13.50	8.04	631.00	11.70	0.95	56.30	1.91
Enwiki	16	3	5.95	793.00	24.90	5.37	417.00	19.20	0.59	37.80	5.67
Enwiki	16	4	4.56	602.00	47.90	4.15	330.00	32.30	0.41	27.50	15.70
Enwiki	32	1	17.40	2540.00	27.60	15.70	1230.00	22.70	1.72	131.00	4.93
Enwiki	32	2	8.82	1300.00	99.10	8.06	664.00	63.90	0.76	63.60	35.40
Enwiki	32	3	6.14	860.00	188.00	5.48	457.00	87.90	0.66	40.50	101.00
Enwiki	32	4	5.28	627.00	222.00	4.20	334.00	92.60	1.09	29.50	130.00
Ruwiki	4	1	38.10	4460.00	36.80	25.60	2130.00	36.70	7.03	131.00	0.10
Ruwiki	4	2	18.50	2420.00	18.10	14.70	1380.00	17.80	2.15	58.50	0.21
Ruwiki	4	3	12.30	1660.00	11.80	9.26	881.00	11.50	1.70	43.80	0.20
Ruwiki	4	4	9.39	1280.00	8.82	7.61	711.00	8.75	1.01	32.30	0.04
Ruwiki	16	1				26.00	1740.00	37.40			
Ruwiki	16	2	14.70	1670.00	21.30	13.30	930.00	19.40	0.80	41.90	1.10
Ruwiki	16	3	9.78	1130.00	53.70	8.86	617.00	29.10	0.52	29.10	13.90
Ruwiki	16	4	7.59	865.00	88.80	6.83	486.00	45.10	0.43	21.30	24.60
Ruwiki	32	1	28.00	3170.00	43.70	25.80	1680.00	39.30	1.23	83.70	2.46
Ruwiki	32	2	14.30	1650.00	173.00	13.20	877.00	86.00	0.59	43.80	49.10
Ruwiki	32	3	9.87	1130.00	291.00	8.89	597.00	114.00	0.55	30.10	99.80
Ruwiki	32	4	8.06	833.00	306.00	6.82	451.00	124.00	0.70	21.60	103.00
Zhwiki	4	1	27.80	3310.00	11.50	23.80	2630.00	11.60	7.18	122.00	-0.13
Zhwiki	4	2	13.90	1700.00	5.59	12.40	1360.00	5.52	2.64	60.10	0.12
Zhwiki	4	3	9.43	1150.00	3.69	8.54	921.00	3.65	1.59	41.40	0.07
Zhwiki	4	4	7.44	874.00	2.79	6.57	701.00	2.79	1.54	31.10	-0.00
Zhwiki	16	1	23.80	1840.00	11.90	22.60	1410.00	11.90	2.03	77.30	-0.03
Zhwiki	16	2	12.00	942.00	9.56	11.50	722.00	7.84	0.87	39.50	3.08
Zhwiki	16	3	8.14	634.00	27.60	7.84	491.00	16.70	0.54	25.60	19.50
Zhwiki	16	4	6.33	483.00	45.30	5.98	370.00	23.40	0.62	20.30	39.10
Zhwiki	32	1	23.00	1700.00	15.00	22.30	1260.00	15.30	1.23	78.80	-0.47
Zhwiki	32	2	11.80	884.00	94.40	11.40	629.00	43.90	0.68	45.60	90.50
Zhwiki	32	3	8.05	590.00	128.00	7.75	451.00	54.40	0.54	25.00	133.00
Zhwiki	32	4	6.21	447.00	124.00	5.90	329.00	51.70	0.57	21.00	130.00
								μ	**1.51**	49.10	27.50
								σ	1.73	33.20	41.50

Table D.70: `gb18030` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	25.00	101.00	33.50	14.10	66.10	34.70	4.93	1.56	-0.53
Arwiki	4	2	17.40	718.00	24.90	10.30	382.00	20.30	3.21	15.20	2.06
Arwiki	4	3	12.70	569.00	18.30	8.44	335.00	14.60	1.92	10.60	1.68
Arwiki	4	4	10.50	466.00	14.50	6.45	262.00	11.20	1.81	9.21	1.51
Arwiki	16	1	20.00	937.00	35.20	14.60	348.00	35.20	2.41	26.60	0.03
Arwiki	16	2	10.80	592.00	22.50	8.47	296.00	23.20	1.05	13.40	-0.31
Arwiki	16	3	7.41	403.00	20.70	5.86	209.00	21.50	0.70	8.75	-0.40
Arwiki	16	4	5.97	318.00	25.00	4.66	169.00	23.20	0.59	6.72	0.79
Arwiki	32	1	19.80	1150.00	40.50	15.30	458.00	37.20	2.01	31.10	1.47
Arwiki	32	2	9.88	611.00	68.20	8.29	280.00	49.10	0.72	14.90	8.65
Arwiki	32	3	7.37	406.00	87.70	5.69	185.00	49.40	0.76	9.98	17.30
Arwiki	32	4	8.35	309.00	91.90	4.63	145.00	43.80	1.68	7.39	21.70
Enwiki	4	1	13.90	77.80	15.60	8.18	29.60	15.60	5.76	4.85	0.06
Enwiki	4	2	9.12	356.00	10.30	6.20	218.00	9.37	2.94	13.80	0.97
Enwiki	4	3	6.33	270.00	7.32	4.09	157.00	6.43	2.26	11.40	0.90
Enwiki	4	4	5.18	222.00	5.82	3.53	135.00	5.33	1.67	8.69	0.50
Enwiki	16	1				9.71	302.00	15.90			
Enwiki	16	2	6.70	477.00	9.32	5.29	196.00	9.66	1.42	28.30	-0.34
Enwiki	16	3	4.58	321.00	10.40	3.59	132.00	10.20	1.00	19.10	0.22
Enwiki	16	4	3.65	240.00	15.50	2.88	111.00	11.80	0.77	13.00	3.73
Enwiki	32	1	13.20	921.00	20.10	10.30	386.00	17.40	2.90	53.90	2.67
Enwiki	32	2	6.36	477.00	42.80	5.30	209.00	28.80	1.07	26.90	14.10
Enwiki	32	3	4.88	317.00	58.80	3.66	138.00	32.00	1.23	18.10	26.90
Enwiki	32	4	6.35	241.00	68.50	3.13	107.00	29.40	3.24	13.50	39.40
Ruwiki	4	1	23.70	104.00	27.30	14.20	53.60	27.90	5.32	2.84	-0.36
Ruwiki	4	2	16.00	649.00	20.00	11.20	406.00	16.90	2.73	13.70	1.79
Ruwiki	4	3	11.30	507.00	14.80	7.43	292.00	11.80	2.20	12.10	1.70
Ruwiki	4	4	9.36	421.00	11.90	6.40	253.00	9.74	1.67	9.50	1.19
Ruwiki	16	1	20.90	1130.00	28.40	15.50	399.00	28.20	3.05	41.00	0.10
Ruwiki	16	2	10.60	640.00	16.70	8.59	291.00	17.70	1.12	19.60	-0.59
Ruwiki	16	3	7.23	432.00	17.60	5.84	198.00	17.60	0.78	13.20	-0.03
Ruwiki	16	4	5.81	336.00	25.60	4.66	164.00	20.30	0.65	9.66	2.98
Ruwiki	32	1	20.10	1250.00	34.50	16.00	508.00	31.20	2.34	41.90	1.88
Ruwiki	32	2	9.88	647.00	75.00	8.35	288.00	44.50	0.86	20.20	17.20
Ruwiki	32	3	7.80	431.00	100.00	5.72	189.00	48.30	1.17	13.60	29.30
Ruwiki	32	4	9.74	327.00	106.00	4.76	148.00	45.40	2.80	10.10	34.30
Zhwiki	4	1	18.90	418.00	8.80	15.00	184.00	8.82	6.97	41.90	-0.03
Zhwiki	4	2	10.20	358.00	8.76	8.51	188.00	6.24	3.06	30.40	4.51
Zhwiki	4	3	7.02	257.00	6.71	5.99	140.00	4.86	1.85	21.00	3.32
Zhwiki	4	4	5.52	198.00	5.08	4.64	110.00	3.69	1.57	15.80	2.48
Zhwiki	16	1	17.50	730.00	8.91	15.50	345.00	8.85	3.58	69.00	0.12
Zhwiki	16	2	8.62	374.00	7.06	7.91	190.00	6.75	1.28	33.00	0.56
Zhwiki	16	3	5.89	252.00	10.80	5.41	131.00	8.66	0.87	21.80	3.83
Zhwiki	16	4	4.58	190.00	15.30	4.15	101.00	10.10	0.76	16.00	9.42
Zhwiki	32	1	16.40	618.00	14.30	15.40	353.00	12.00	1.83	47.60	4.05
Zhwiki	32	2	8.17	314.00	37.90	7.76	184.00	19.40	0.75	23.10	33.20
Zhwiki	32	3	5.66	210.00	46.30	5.31	125.00	21.10	0.63	15.20	45.20
Zhwiki	32	4	4.94	157.00	46.60	4.12	94.80	19.90	1.46	11.20	47.90
								μ	**2.03**	19.60	8.24
								σ	1.42	13.80	13.10

Table D.71: `gb18030` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.5 Case study: u8u16

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	6.89	5.03	2.40	6.60	5.93	3.05	0.20	-0.06	-0.46
Arwiki	4	2	3.79	8.16	.90	3.54	8.96	.84	0.18	-0.06	0.04
Arwiki	4	3	3.42	7.39	.64	2.52	8.18	.71	0.63	-0.06	-0.05
Arwiki	4	4	3.88	4.60	.31	2.28	8.03	.38	1.12	-0.24	-0.05
Arwiki	16	1	6.56	8.99	2.54	6.52	9.76	2.96	0.02	-0.05	-0.29
Arwiki	16	2	3.39	6.10	.98	3.39	6.25	.95	0.00	-0.01	0.02
Arwiki	16	3	3.37	4.56	.72	2.22	4.53	.85	0.80	0.00	-0.09
Arwiki	16	4	3.72	3.76	.33	1.82	3.72	.36	1.33	0.00	-0.02
Arwiki	32	1	6.55	24.20	2.89	6.49	15.30	2.30	0.04	0.62	0.41
Arwiki	32	2	3.42	11.50	1.04	3.35	9.81	.96	0.05	0.11	0.06
Arwiki	32	3	3.47	8.99	1.12	2.17	5.47	.72	0.91	0.24	0.28
Arwiki	32	4	3.73	7.09	.71	1.71	4.65	.45	1.41	0.17	0.18
Enwiki	4	1	4.82	3.35	1.54	4.68	4.03	2.15	0.14	-0.07	-0.61
Enwiki	4	2	2.59	5.51	.64	2.48	7.09	.62	0.12	-0.16	0.02
Enwiki	4	3	2.39	4.53	.54	1.77	6.72	.48	0.63	-0.22	0.06
Enwiki	4	4	2.63	3.18	.24	1.61	6.94	.27	1.03	-0.38	-0.03
Enwiki	16	1	4.61	5.90	1.72	4.59	7.28	1.85	0.02	-0.14	-0.13
Enwiki	16	2	2.38	4.06	.66	2.39	6.59	.71	-0.01	-0.26	-0.06
Enwiki	16	3	2.37	3.19	.46	1.71	6.59	.54	0.67	-0.34	-0.07
Enwiki	16	4	2.62	2.95	.25	1.30	4.02	.27	1.33	-0.11	-0.02
Enwiki	32	1	4.62	19.60	2.14	4.58	12.90	1.69	0.04	0.68	0.46
Enwiki	32	2	2.46	9.77	.90	2.39	10.50	.75	0.07	-0.08	0.15
Enwiki	32	3	2.43	7.26	.88	1.57	6.90	.58	0.87	0.04	0.31
Enwiki	32	4	2.58	5.75	.58	1.22	4.91	.41	1.37	0.08	0.18
Ruwiki	4	1	8.71	5.86	2.59	8.34	7.16	3.64	0.21	-0.07	-0.59
Ruwiki	4	2	4.61	10.70	1.07	4.50	11.30	1.07	0.07	-0.04	-0.00
Ruwiki	4	3	4.53	8.00	.76	3.23	10.40	.87	0.74	-0.14	-0.07
Ruwiki	4	4	4.68	6.10	.42	3.00	12.70	.49	0.95	-0.37	-0.04
Ruwiki	16	1	8.29	10.50	3.25	8.25	11.60	3.82	0.02	-0.06	-0.33
Ruwiki	16	2	4.29	6.67	1.13	4.32	8.55	1.20	-0.02	-0.11	-0.04
Ruwiki	16	3	4.18	5.56	.85	2.77	4.98	1.03	0.80	0.03	-0.10
Ruwiki	16	4	4.73	4.62	.40	2.32	5.16	.46	1.36	-0.03	-0.04
Ruwiki	32	1	8.24	28.70	3.23	8.17	16.80	2.67	0.04	0.67	0.32
Ruwiki	32	2	4.39	13.70	1.27	4.23	15.00	1.16	0.09	-0.08	0.06
Ruwiki	32	3	4.20	10.00	1.32	2.73	6.26	.85	0.83	0.21	0.27
Ruwiki	32	4	4.72	8.84	.90	2.13	5.43	.52	1.47	0.19	0.22
Zhwiki	4	1	3.43	2.23	.99	3.28	2.69	1.36	0.21	-0.07	-0.54
Zhwiki	4	2	1.82	4.12	.40	1.77	4.62	.41	0.07	-0.07	-0.00
Zhwiki	4	3	1.63	3.49	.30	1.29	4.29	.34	0.51	-0.12	-0.06
Zhwiki	4	4	1.75	2.49	.17	1.35	6.78	.19	0.59	-0.63	-0.03
Zhwiki	16	1	3.28	3.97	1.30	3.25	4.49	1.45	0.03	-0.08	-0.23
Zhwiki	16	2	1.70	2.52	.43	1.73	3.60	.45	-0.05	-0.16	-0.03
Zhwiki	16	3	1.63	2.11	.32	1.10	1.95	.41	0.79	0.02	-0.13
Zhwiki	16	4	1.77	1.69	.16	1.50	7.73	.17	0.39	-0.89	-0.01
Zhwiki	32	1	3.25	10.30	1.32	3.22	6.53	1.02	0.04	0.55	0.43
Zhwiki	32	2	1.69	4.91	.54	1.69	4.27	.52	-0.01	0.09	0.04
Zhwiki	32	3	1.61	3.78	.50	1.08	2.37	.37	0.78	0.21	0.20
Zhwiki	32	4	1.74	2.95	.32	.97	3.19	.21	1.13	-0.04	0.17
								μ	**0.50**	-0.03	-0.01
								σ	0.49	0.28	0.24

Table D.72: u8u16 PAPI comparison between 2020 and CURR on SSE-4.2

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	5.40	48.70	31.60	5.09	57.50	31.60	0.22	-0.62	-0.04
Arwiki	4	2	2.91	69.90	15.90	2.72	64.60	15.80	0.13	0.37	0.07
Arwiki	4	3	2.77	56.80	10.50	1.92	54.20	10.40	0.59	0.18	0.05
Arwiki	4	4	2.86	46.20	7.84	1.67	45.00	7.82	0.83	0.08	0.01
Arwiki	16	1	5.14	90.00	31.70	5.05	99.50	31.80	0.06	-0.67	-0.06
Arwiki	16	2	2.73	68.40	15.80	2.62	61.50	15.90	0.08	0.48	-0.06
Arwiki	16	3	2.68	50.50	10.40	1.75	45.70	10.40	0.65	0.34	-0.03
Arwiki	16	4	2.72	40.70	7.74	1.48	36.20	7.88	0.86	0.31	-0.10
Arwiki	32	1	5.14	243.00	30.60	5.06	181.00	31.70	0.05	4.33	-0.73
Arwiki	32	2	2.66	119.00	15.20	2.62	98.20	15.80	0.03	1.46	-0.43
Arwiki	32	3	2.65	86.70	9.96	1.71	63.00	10.30	0.66	1.66	-0.26
Arwiki	32	4	2.72	65.20	7.50	1.44	48.20	7.84	0.89	1.19	-0.24
Enwiki	4	1	3.79	34.20	21.80	3.60	37.80	22.00	0.19	-0.36	-0.22
Enwiki	4	2	2.11	35.20	11.00	1.92	54.40	10.90	0.19	-1.94	0.09
Enwiki	4	3	1.99	28.50	7.30	1.37	48.10	7.25	0.62	-1.98	0.05
Enwiki	4	4	2.01	24.20	5.44	1.20	39.70	5.41	0.82	-1.55	0.04
Enwiki	16	1	3.60	71.50	21.90	3.55	68.60	22.10	0.05	0.29	-0.21
Enwiki	16	2	1.90	41.10	11.00	1.83	54.90	11.00	0.06	-1.39	-0.08
Enwiki	16	3	1.89	30.80	7.19	1.32	46.50	7.20	0.57	-1.58	-0.01
Enwiki	16	4	1.99	28.20	5.36	1.07	32.20	5.45	0.92	-0.40	-0.09
Enwiki	32	1	3.59	188.00	21.10	3.55	152.00	21.90	0.04	3.64	-0.87
Enwiki	32	2	1.86	97.70	10.40	1.82	86.80	10.90	0.04	1.10	-0.46
Enwiki	32	3	1.88	70.50	6.89	1.25	56.10	7.15	0.63	1.46	-0.27
Enwiki	32	4	2.01	58.00	5.20	1.05	41.40	5.41	0.97	1.67	-0.21
Ruwiki	4	1	6.79	60.30	38.90	6.38	69.00	39.40	0.23	-0.49	-0.28
Ruwiki	4	2	3.64	75.20	19.60	3.41	77.30	19.50	0.13	-0.12	0.07
Ruwiki	4	3	3.42	61.60	12.90	2.42	64.80	12.90	0.57	-0.18	0.04
Ruwiki	4	4	3.50	51.20	9.65	2.10	55.40	9.62	0.80	-0.24	0.02
Ruwiki	16	1	6.44	108.00	39.00	6.32	121.00	39.40	0.07	-0.75	-0.20
Ruwiki	16	2	3.38	74.10	19.50	3.28	73.30	19.60	0.06	0.04	-0.05
Ruwiki	16	3	3.29	56.00	12.80	2.17	52.50	12.80	0.64	0.20	-0.04
Ruwiki	16	4	3.38	45.00	9.59	1.78	41.40	9.71	0.91	0.20	-0.07
Ruwiki	32	1	6.40	284.00	37.80	6.31	214.00	39.30	0.05	3.95	-0.89
Ruwiki	32	2	3.32	142.00	18.80	3.24	114.00	19.60	0.04	1.54	-0.44
Ruwiki	32	3	3.27	103.00	12.30	2.13	69.20	12.80	0.64	1.91	-0.27
Ruwiki	32	4	3.33	77.00	9.23	1.74	53.80	9.68	0.90	1.31	-0.26
Zhwiki	4	1	2.63	23.30	15.00	2.47	26.80	15.10	0.23	-0.51	-0.08
Zhwiki	4	2	1.42	30.70	7.55	1.32	30.00	7.50	0.15	0.10	0.06
Zhwiki	4	3	1.38	24.10	5.03	.94	25.10	4.97	0.64	-0.15	0.09
Zhwiki	4	4	1.37	20.60	3.74	.84	21.90	3.72	0.77	-0.20	0.03
Zhwiki	16	1	2.50	43.30	15.00	2.45	45.90	15.10	0.07	-0.39	-0.16
Zhwiki	16	2	1.38	31.50	7.58	1.27	29.40	7.57	0.17	0.31	0.02
Zhwiki	16	3	1.34	24.50	4.96	.84	19.90	4.95	0.74	0.68	0.00
Zhwiki	16	4	1.33	19.00	3.70	.71	15.60	3.75	0.91	0.50	-0.07
Zhwiki	32	1	2.48	108.00	14.50	2.45	83.70	15.10	0.04	3.60	-0.79
Zhwiki	32	2	1.38	59.90	7.26	1.27	42.00	7.53	0.16	2.62	-0.41
Zhwiki	32	3	1.29	41.90	4.77	.82	27.60	4.93	0.69	2.11	-0.22
Zhwiki	32	4	1.37	34.30	3.58	.70	20.50	3.74	0.99	2.03	-0.23
								μ	**0.43**	0.54	-0.17
								σ	0.34	1.45	0.24

Table D.73: `u8u16` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	3.13	22.90	22.40	2.80	22.90	22.40	0.23	-0.00	0.00
Arwiki	4	2	1.88	34.40	11.20	1.69	29.80	11.20	0.14	0.32	-0.00
Arwiki	4	3	1.54	28.30	7.48	1.29	24.30	7.47	0.18	0.28	0.01
Arwiki	4	4	1.52	23.10	5.61	1.09	19.30	5.61	0.29	0.26	0.00
Arwiki	16	1	2.99	22.80	22.40	2.89	22.90	22.40	0.07	-0.00	-0.00
Arwiki	16	2	1.61	25.00	11.20	1.57	19.70	11.20	0.03	0.37	0.01
Arwiki	16	3	1.38	19.00	7.47	1.13	19.40	7.47	0.17	-0.03	0.00
Arwiki	16	4	1.42	15.90	5.63	.92	15.30	5.60	0.35	0.04	0.02
Arwiki	32	1	2.94	23.00	22.40	2.88	23.30	22.40	0.04	-0.02	-0.00
Arwiki	32	2	1.56	21.00	11.20	1.53	19.60	11.30	0.02	0.09	-0.04
Arwiki	32	3	1.35	16.60	7.51	1.01	17.80	7.49	0.23	-0.09	0.01
Arwiki	32	4	1.38	13.90	5.65	.83	14.60	5.62	0.38	-0.06	0.02
Enwiki	4	1	2.20	15.80	15.50	1.99	15.90	15.50	0.21	-0.00	-0.00
Enwiki	4	2	1.27	18.60	7.76	1.18	19.60	7.76	0.10	-0.10	0.01
Enwiki	4	3	1.03	14.90	5.18	.88	21.40	5.18	0.15	-0.65	0.01
Enwiki	4	4	1.01	12.00	3.89	.74	17.00	3.89	0.26	-0.50	-0.00
Enwiki	16	1	2.09	15.80	15.50	2.04	16.10	15.50	0.05	-0.02	-0.00
Enwiki	16	2	1.12	11.80	7.76	1.11	20.70	7.76	0.01	-0.89	-0.00
Enwiki	16	3	.95	8.46	5.18	.90	19.60	5.17	0.05	-1.12	0.00
Enwiki	16	4	.96	6.79	3.90	.70	13.90	3.88	0.27	-0.72	0.02
Enwiki	32	1	2.05	15.90	15.50	2.03	16.50	15.50	0.02	-0.06	-0.00
Enwiki	32	2	1.09	9.17	7.76	1.08	18.60	7.87	0.00	-0.95	-0.11
Enwiki	32	3	.93	6.57	5.18	.82	18.00	5.20	0.10	-1.15	-0.01
Enwiki	32	4	.94	5.35	3.91	.66	13.20	3.91	0.28	-0.79	-0.00
Ruwiki	4	1	3.91	28.20	27.60	3.50	28.20	27.60	0.23	-0.00	0.00
Ruwiki	4	2	2.34	37.50	13.80	2.11	36.00	13.80	0.13	0.09	-0.00
Ruwiki	4	3	1.90	30.80	9.21	1.64	29.10	9.20	0.15	0.10	0.00
Ruwiki	4	4	1.88	25.30	6.92	1.37	23.50	6.91	0.29	0.10	0.01
Ruwiki	16	1	3.73	28.10	27.60	3.62	28.20	27.60	0.06	-0.01	0.00
Ruwiki	16	2	2.02	26.40	13.80	1.96	25.50	13.80	0.03	0.05	0.00
Ruwiki	16	3	1.69	19.80	9.21	1.43	23.70	9.20	0.15	-0.22	0.01
Ruwiki	16	4	1.73	16.50	6.93	1.14	17.70	6.90	0.34	-0.07	0.02
Ruwiki	32	1	3.66	28.30	27.60	3.61	28.70	27.60	0.03	-0.03	-0.00
Ruwiki	32	2	1.94	21.60	13.80	1.91	28.60	13.80	0.02	-0.40	0.01
Ruwiki	32	3	1.64	16.40	9.24	1.24	20.90	9.22	0.23	-0.25	0.01
Ruwiki	32	4	1.68	13.70	6.95	1.03	16.90	6.92	0.37	-0.18	0.02
Zhwiki	4	1	1.51	10.90	10.60	1.35	10.90	10.60	0.23	-0.00	0.00
Zhwiki	4	2	.91	15.30	5.33	.83	13.90	5.33	0.12	0.20	-0.00
Zhwiki	4	3	.73	12.30	3.55	.63	11.10	3.55	0.15	0.17	0.00
Zhwiki	4	4	.73	10.30	2.68	.53	9.00	2.67	0.29	0.19	0.02
Zhwiki	16	1	1.44	10.90	10.60	1.40	10.90	10.60	0.06	-0.01	-0.00
Zhwiki	16	2	.78	11.00	5.33	.76	10.40	5.33	0.03	0.09	-0.00
Zhwiki	16	3	.66	8.31	3.56	.56	8.73	3.55	0.14	-0.06	0.01
Zhwiki	16	4	.67	7.01	2.69	.44	6.64	2.66	0.34	0.06	0.04
Zhwiki	32	1	1.42	10.90	10.60	1.40	11.30	10.60	0.03	-0.05	-0.00
Zhwiki	32	2	.75	9.13	5.35	.73	10.60	5.34	0.02	-0.21	0.01
Zhwiki	32	3	.64	7.01	3.57	.48	7.60	3.56	0.23	-0.09	0.01
Zhwiki	32	4	.65	5.90	2.70	.40	6.23	2.67	0.38	-0.05	0.04
									$\boldsymbol{\mu}$	**0.16**	-0.13	0.00
									$\boldsymbol{\sigma}$	0.12	0.36	0.02

Table D.74: `u8u16` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.6 Case study: u32u8

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	32.10	85.10	13.40	29.90	74.70	12.60	0.54	0.25	0.19
Arwiki	4	2	17.20	62.30	6.50	15.40	62.70	5.63	0.43	-0.01	0.21
Arwiki	4	3	12.20	43.90	4.12	10.20	43.60	3.74	0.46	0.01	0.09
Arwiki	4	4	10.80	32.90	3.09	8.12	35.00	2.87	0.65	-0.05	0.05
Arwiki	16	1	29.80	96.20	18.00	29.40	91.40	18.00	0.11	0.11	0.00
Arwiki	16	2	15.60	51.20	8.96	14.70	50.20	9.09	0.21	0.02	-0.03
Arwiki	16	3	11.20	34.80	5.95	9.85	31.60	6.89	0.33	0.08	-0.23
Arwiki	16	4	9.67	27.00	4.42	7.56	27.20	6.45	0.51	-0.01	-0.49
Arwiki	32	1	29.30	108.00	16.10	29.00	99.60	16.20	0.05	0.20	-0.04
Arwiki	32	2	15.40	53.80	8.19	14.60	55.80	11.00	0.17	-0.05	-0.67
Arwiki	32	3	11.00	36.90	6.24	9.95	36.80	9.62	0.26	0.00	-0.81
Arwiki	32	4	10.40	30.90	5.43	7.70	28.80	7.92	0.65	0.05	-0.60
Enwiki	4	1	24.40	77.20	15.60	22.40	79.10	15.10	0.49	-0.05	0.12
Enwiki	4	2	12.80	55.60	7.12	11.70	58.00	7.29	0.26	-0.06	-0.04
Enwiki	4	3	8.95	41.50	4.81	7.62	41.90	4.49	0.34	-0.01	0.08
Enwiki	4	4	8.17	29.50	3.13	5.99	32.20	3.46	0.55	-0.07	-0.09
Enwiki	16	1	22.40	88.90	17.50	22.00	83.30	17.20	0.11	0.14	0.08
Enwiki	16	2	11.80	47.20	8.73	11.10	45.60	8.78	0.18	0.04	-0.01
Enwiki	16	3	8.17	32.20	5.80	7.38	28.90	6.64	0.20	0.08	-0.21
Enwiki	16	4	7.61	24.50	4.31	5.63	24.80	6.36	0.50	-0.01	-0.52
Enwiki	32	1	21.90	95.70	16.30	21.70	89.20	16.60	0.06	0.17	-0.09
Enwiki	32	2	11.30	48.30	8.46	11.00	49.20	11.00	0.08	-0.02	-0.64
Enwiki	32	3	8.27	33.30	6.28	7.47	32.50	9.47	0.20	0.02	-0.81
Enwiki	32	4	7.51	27.00	5.12	5.80	25.00	7.46	0.43	0.05	-0.59
Ruwiki	4	1	38.10	100.00	14.20	35.60	86.30	13.30	0.56	0.31	0.20
Ruwiki	4	2	19.80	72.90	6.79	18.40	71.20	6.01	0.31	0.04	0.17
Ruwiki	4	3	14.00	50.40	4.60	12.20	49.30	3.94	0.40	0.02	0.14
Ruwiki	4	4	13.20	36.80	3.42	9.50	39.90	3.09	0.81	-0.07	0.07
Ruwiki	16	1	35.60	110.00	19.70	35.00	105.00	19.70	0.13	0.10	-0.00
Ruwiki	16	2	18.60	57.90	9.78	17.60	57.50	9.96	0.23	0.01	-0.04
Ruwiki	16	3	14.00	39.90	6.44	11.80	35.70	7.25	0.49	0.09	-0.18
Ruwiki	16	4	12.00	31.00	4.88	8.92	30.80	7.05	0.67	0.00	-0.48
Ruwiki	32	1	34.90	128.00	17.20	34.60	119.00	17.60	0.07	0.21	-0.07
Ruwiki	32	2	18.90	63.80	9.07	17.50	65.20	11.80	0.31	-0.03	-0.60
Ruwiki	32	3	12.90	42.60	7.14	11.90	42.30	11.00	0.23	0.01	-0.85
Ruwiki	32	4	11.50	35.10	6.19	8.94	32.80	8.44	0.57	0.05	-0.49
Zhwiki	4	1	14.70	39.00	5.47	13.70	33.20	5.12	0.55	0.33	0.20
Zhwiki	4	2	7.50	28.30	2.65	7.07	26.70	2.26	0.25	0.09	0.22
Zhwiki	4	3	5.25	19.60	1.74	4.77	19.20	1.58	0.28	0.02	0.09
Zhwiki	4	4	5.00	14.20	1.33	3.84	15.00	1.20	0.67	-0.05	0.08
Zhwiki	16	1	13.70	42.70	7.51	13.50	40.00	7.50	0.14	0.15	0.01
Zhwiki	16	2	7.01	22.30	3.73	6.80	22.10	3.77	0.12	0.01	-0.03
Zhwiki	16	3	5.12	15.40	2.47	4.59	14.00	2.82	0.30	0.08	-0.20
Zhwiki	16	4	4.50	11.90	1.85	3.57	12.20	2.84	0.54	-0.01	-0.57
Zhwiki	32	1	13.50	51.00	6.72	13.30	46.20	6.74	0.08	0.28	-0.01
Zhwiki	32	2	6.96	24.70	3.45	6.77	25.70	4.48	0.11	-0.06	-0.60
Zhwiki	32	3	5.07	16.70	2.73	4.66	16.70	4.27	0.24	-0.00	-0.89
Zhwiki	32	4	4.70	13.60	2.37	3.53	13.00	3.48	0.68	0.03	-0.65
								μ	**0.34**	0.05	-0.20
								σ	0.20	0.10	0.34

Table D.75: u32u8 PAPI comparison between 2020 and CURR on SSE-4.2

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	10.20	670.00	82.20	8.04	477.00	80.90	0.51	4.62	0.32
Arwiki	4	2	5.17	460.00	40.50	4.45	356.00	40.60	0.17	2.51	-0.02
Arwiki	4	3	3.47	327.00	26.90	3.05	251.00	26.90	0.10	1.81	0.00
Arwiki	4	4	2.86	248.00	20.00	2.44	195.00	20.30	0.10	1.28	-0.07
Arwiki	16	1	8.59	786.00	79.10	7.99	548.00	79.50	0.14	5.69	-0.08
Arwiki	16	2	4.26	410.00	39.60	4.03	294.00	39.80	0.06	2.78	-0.04
Arwiki	16	3	2.85	279.00	26.50	2.70	201.00	27.80	0.04	1.90	-0.32
Arwiki	16	4	2.20	212.00	20.20	2.05	151.00	25.40	0.04	1.45	-1.26
Arwiki	32	1	8.35	862.00	79.00	8.02	645.00	79.10	0.08	5.21	-0.02
Arwiki	32	2	4.15	439.00	41.50	4.01	325.00	50.60	0.04	2.75	-2.18
Arwiki	32	3	2.80	299.00	40.60	2.74	227.00	63.00	0.01	1.72	-5.37
Arwiki	32	4	2.26	224.00	51.80	2.11	168.00	61.40	0.04	1.34	-2.31
Enwiki	4	1	8.08	501.00	77.90	6.27	398.00	76.40	0.46	2.60	0.37
Enwiki	4	2	4.20	366.00	38.20	3.69	284.00	38.40	0.13	2.05	-0.05
Enwiki	4	3	2.80	263.00	25.30	2.36	208.00	25.30	0.11	1.37	0.01
Enwiki	4	4	2.35	199.00	19.00	1.99	159.00	19.20	0.09	1.01	-0.04
Enwiki	16	1	6.75	612.00	75.30	6.25	448.00	75.50	0.13	4.15	-0.05
Enwiki	16	2	3.35	325.00	37.60	3.16	246.00	37.70	0.05	1.98	-0.02
Enwiki	16	3	2.24	222.00	25.10	2.12	166.00	26.10	0.03	1.41	-0.26
Enwiki	16	4	1.81	168.00	19.40	1.63	126.00	23.50	0.04	1.07	-1.04
Enwiki	32	1	6.57	662.00	75.00	6.30	509.00	75.50	0.07	3.87	-0.12
Enwiki	32	2	3.27	341.00	38.80	3.15	260.00	45.80	0.03	2.06	-1.78
Enwiki	32	3	2.23	232.00	34.70	2.16	177.00	52.50	0.02	1.38	-4.51
Enwiki	32	4	1.88	173.00	40.60	1.67	130.00	50.10	0.05	1.08	-2.40
Ruwiki	4	1	11.80	787.00	89.90	9.39	550.00	88.50	0.53	5.23	0.31
Ruwiki	4	2	6.03	534.00	44.00	5.20	411.00	44.20	0.18	2.70	-0.03
Ruwiki	4	3	3.98	381.00	29.30	3.49	289.00	29.30	0.11	2.03	0.01
Ruwiki	4	4	3.19	291.00	21.80	2.82	224.00	22.20	0.08	1.46	-0.09
Ruwiki	16	1	10.00	929.00	86.30	9.36	646.00	86.80	0.14	6.23	-0.10
Ruwiki	16	2	4.95	484.00	43.00	4.71	342.00	43.40	0.05	3.11	-0.10
Ruwiki	16	3	3.31	329.00	28.90	3.16	234.00	30.60	0.03	2.10	-0.36
Ruwiki	16	4	2.61	252.00	23.10	2.39	178.00	30.00	0.05	1.63	-1.53
Ruwiki	32	1	9.75	1040.00	86.20	9.38	776.00	86.50	0.08	5.73	-0.06
Ruwiki	32	2	4.94	531.00	44.30	4.69	389.00	58.80	0.06	3.12	-3.19
Ruwiki	32	3	3.25	354.00	45.80	3.21	278.00	73.20	0.01	1.67	-6.02
Ruwiki	32	4	2.53	265.00	61.50	2.47	206.00	71.00	0.01	1.30	-2.08
Zhwiki	4	1	4.53	304.00	34.10	3.61	214.00	33.70	0.53	5.19	0.24
Zhwiki	4	2	2.31	207.00	16.80	2.00	156.00	16.80	0.18	2.93	0.02
Zhwiki	4	3	1.60	146.00	11.20	1.36	111.00	11.10	0.14	2.01	0.04
Zhwiki	4	4	1.40	112.00	8.47	1.09	86.20	8.42	0.18	1.49	0.03
Zhwiki	16	1	3.84	357.00	32.90	3.59	249.00	33.10	0.14	6.25	-0.12
Zhwiki	16	2	1.91	187.00	16.40	1.81	133.00	16.50	0.06	3.10	-0.06
Zhwiki	16	3	1.38	129.00	11.20	1.22	90.30	11.40	0.09	2.25	-0.12
Zhwiki	16	4	1.13	99.90	8.66	.95	68.30	10.50	0.11	1.83	-1.05
Zhwiki	32	1	3.75	401.00	32.90	3.60	299.00	32.90	0.08	5.86	-0.04
Zhwiki	32	2	1.94	206.00	17.10	1.80	148.00	19.60	0.08	3.33	-1.40
Zhwiki	32	3	1.42	143.00	17.20	1.24	104.00	26.60	0.10	2.26	-5.43
Zhwiki	32	4	1.12	108.00	23.10	.99	77.50	27.00	0.07	1.77	-2.24
								μ	**0.12**	2.74	-0.93
								σ	0.13	1.53	1.59

Table D.76: `u32u8` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	7.44	67.30	65.10	5.37	70.10	65.10	0.50	-0.07	0.00
Arwiki	4	2	4.60	175.00	33.80	3.55	144.00	33.70	0.25	0.72	0.02
Arwiki	4	3	3.27	134.00	22.60	2.59	111.00	22.90	0.16	0.53	-0.08
Arwiki	4	4	2.74	112.00	17.20	2.17	88.60	17.40	0.14	0.56	-0.04
Arwiki	16	1	6.21	240.00	65.10	5.51	187.00	65.10	0.17	1.28	-0.00
Arwiki	16	2	3.26	146.00	35.60	2.92	114.00	34.80	0.08	0.77	0.20
Arwiki	16	3	2.25	102.00	24.10	2.03	79.50	24.10	0.05	0.54	0.01
Arwiki	16	4	2.21	77.60	18.20	1.60	60.70	19.20	0.15	0.41	-0.24
Arwiki	32	1	6.16	281.00	65.10	5.46	189.00	65.30	0.17	2.21	-0.03
Arwiki	32	2	3.05	150.00	34.60	2.85	105.00	38.10	0.05	1.08	-0.83
Arwiki	32	3	2.17	102.00	26.00	2.03	71.50	32.50	0.03	0.72	-1.57
Arwiki	32	4	2.31	76.50	24.10	1.92	53.60	29.20	0.09	0.55	-1.23
Enwiki	4	1	6.15	64.40	61.90	4.17	66.10	61.90	0.50	-0.04	0.00
Enwiki	4	2	3.87	144.00	32.10	3.04	121.00	32.70	0.21	0.58	-0.15
Enwiki	4	3	2.85	112.00	21.70	2.18	95.00	21.60	0.17	0.44	0.01
Enwiki	4	4	2.41	94.90	16.50	1.91	77.10	16.90	0.13	0.45	-0.09
Enwiki	16	1	5.04	186.00	61.90	4.32	158.00	61.90	0.18	0.72	-0.00
Enwiki	16	2	2.65	114.00	33.00	2.33	98.20	32.80	0.08	0.41	0.06
Enwiki	16	3	2.02	79.20	22.00	1.64	68.10	23.10	0.10	0.28	-0.28
Enwiki	16	4	2.09	59.90	16.60	1.31	52.30	18.50	0.20	0.19	-0.49
Enwiki	32	1	4.87	206.00	61.90	4.30	159.00	62.00	0.14	1.20	-0.01
Enwiki	32	2	2.47	113.00	33.20	2.28	88.60	35.80	0.05	0.61	-0.65
Enwiki	32	3	2.04	75.60	24.50	1.66	60.40	29.10	0.10	0.39	-1.16
Enwiki	32	4	2.16	57.00	21.80	1.64	45.40	25.60	0.13	0.29	-0.96
Ruwiki	4	1	8.63	73.30	71.00	6.27	76.00	71.00	0.52	-0.06	0.00
Ruwiki	4	2	5.17	198.00	36.70	4.12	165.00	36.70	0.23	0.73	0.01
Ruwiki	4	3	3.69	155.00	24.70	2.96	126.00	24.90	0.16	0.63	-0.05
Ruwiki	4	4	3.06	131.00	18.80	2.49	101.00	18.80	0.13	0.66	0.01
Ruwiki	16	1	7.22	283.00	71.00	6.46	216.00	71.00	0.17	1.48	0.00
Ruwiki	16	2	3.76	171.00	37.80	3.39	131.00	37.90	0.08	0.89	-0.03
Ruwiki	16	3	2.58	119.00	25.30	2.34	90.80	25.90	0.05	0.62	-0.13
Ruwiki	16	4	2.40	90.70	19.20	1.83	69.20	20.70	0.12	0.47	-0.32
Ruwiki	32	1	7.25	340.00	71.10	6.36	218.00	71.10	0.20	2.69	-0.01
Ruwiki	32	2	3.55	179.00	36.80	3.29	120.00	40.00	0.06	1.30	-0.71
Ruwiki	32	3	2.47	121.00	26.80	2.35	82.00	35.60	0.03	0.87	-1.94
Ruwiki	32	4	2.50	91.20	24.90	2.18	61.60	32.40	0.07	0.65	-1.63
Zhwiki	4	1	3.33	27.90	27.00	2.42	29.10	27.00	0.53	-0.07	0.00
Zhwiki	4	2	2.01	77.10	14.00	1.59	63.00	14.00	0.24	0.81	-0.01
Zhwiki	4	3	1.44	60.00	9.36	1.14	48.50	9.47	0.17	0.67	-0.06
Zhwiki	4	4	1.21	50.10	7.14	.98	38.90	7.24	0.13	0.65	-0.06
Zhwiki	16	1	2.78	110.00	27.00	2.48	82.60	27.10	0.17	1.61	-0.01
Zhwiki	16	2	1.44	66.40	14.40	1.31	50.40	14.40	0.08	0.93	-0.01
Zhwiki	16	3	1.01	46.00	9.74	.91	34.90	9.88	0.06	0.64	-0.08
Zhwiki	16	4	.92	35.20	7.41	.71	26.60	7.82	0.12	0.50	-0.24
Zhwiki	32	1	2.79	131.00	27.10	2.45	84.00	27.10	0.19	2.72	-0.04
Zhwiki	32	2	1.36	68.70	14.00	1.27	46.20	15.40	0.05	1.30	-0.81
Zhwiki	32	3	.96	46.50	10.40	.91	31.50	13.50	0.03	0.87	-1.81
Zhwiki	32	4	.96	35.30	9.62	.85	23.70	12.50	0.06	0.67	-1.65
								μ	**0.15**	0.77	-0.36
								σ	0.12	0.59	0.57

Table D.77: `u32u8` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.7 Case study: `icgrep -colours=never`

Arguments: @ -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.67	11.30	4.77	1.30	11.80	5.13	0.25	-0.04	-0.25
Arwiki	4	2	1.03	11.20	2.14	.85	13.70	2.16	0.13	-0.18	-0.01
Arwiki	4	3	.83	8.48	1.61	.69	10.90	1.95	0.10	-0.17	-0.24
Arwiki	4	4	.75	6.00	.92	.67	9.35	1.18	0.06	-0.23	-0.18
Arwiki	16	1	1.29	12.50	5.66	1.21	12.80	5.93	0.06	-0.02	-0.19
Arwiki	16	2	.74	8.20	2.79	.73	9.14	2.79	0.00	-0.07	0.00
Arwiki	16	3	.62	5.43	2.01	.58	6.92	2.15	0.03	-0.10	-0.10
Arwiki	16	4	.57	3.56	1.12	.54	5.15	1.40	0.02	-0.11	-0.20
Arwiki	32	1	1.23	12.90	6.02	1.21	13.30	6.27	0.02	-0.02	-0.18
Arwiki	32	2	.77	6.66	2.61	.70	8.53	2.91	0.05	-0.13	-0.21
Arwiki	32	3	.57	4.78	2.09	.55	6.54	2.32	0.01	-0.12	-0.16
Arwiki	32	4	.55	3.15	1.17	.53	4.66	1.51	0.02	-0.10	-0.24
Enwiki	4	1	1.14	7.59	3.12	.89	7.89	3.36	0.25	-0.03	-0.24
Enwiki	4	2	.68	8.33	1.44	.58	9.42	1.44	0.10	-0.11	0.00
Enwiki	4	3	.57	5.81	1.09	.47	7.49	1.31	0.09	-0.17	-0.22
Enwiki	4	4	.50	4.54	.66	.47	6.42	.75	0.03	-0.19	-0.10
Enwiki	16	1	.89	8.61	3.88	.83	8.66	3.97	0.06	-0.01	-0.09
Enwiki	16	2	.54	5.26	1.80	.50	6.25	1.85	0.04	-0.10	-0.05
Enwiki	16	3	.46	3.35	1.25	.40	4.75	1.45	0.06	-0.14	-0.19
Enwiki	16	4	.42	2.47	.75	.36	3.56	.96	0.06	-0.11	-0.21
Enwiki	32	1	.84	8.94	4.13	.82	8.82	4.09	0.02	0.01	0.05
Enwiki	32	2	.49	4.93	1.87	.48	5.83	1.95	0.01	-0.09	-0.08
Enwiki	32	3	.39	3.38	1.47	.38	4.54	1.59	0.01	-0.12	-0.13
Enwiki	32	4	.39	2.29	.85	.36	3.25	1.05	0.03	-0.10	-0.20
Ruwiki	4	1	2.07	14.10	6.02	1.62	14.60	6.23	0.25	-0.03	-0.12
Ruwiki	4	2	1.24	14.30	2.72	1.03	16.90	2.73	0.12	-0.15	-0.01
Ruwiki	4	3	1.06	9.86	1.89	.85	13.50	2.35	0.12	-0.21	-0.27
Ruwiki	4	4	.95	7.22	1.07	.81	11.60	1.42	0.08	-0.25	-0.20
Ruwiki	16	1	1.60	15.60	7.15	1.51	16.10	7.48	0.05	-0.03	-0.19
Ruwiki	16	2	.97	9.24	3.23	.88	11.20	3.42	0.05	-0.11	-0.10
Ruwiki	16	3	.81	6.16	2.30	.70	8.48	2.54	0.06	-0.13	-0.14
Ruwiki	16	4	.70	4.36	1.36	.64	6.32	1.70	0.04	-0.11	-0.19
Ruwiki	32	1	1.54	16.50	7.79	1.48	16.20	7.51	0.03	0.02	0.16
Ruwiki	32	2	.96	8.42	3.28	.87	10.50	3.51	0.05	-0.12	-0.13
Ruwiki	32	3	.73	5.78	2.53	.67	8.06	2.79	0.03	-0.13	-0.14
Ruwiki	32	4	.69	3.95	1.47	.66	5.64	1.84	0.02	-0.10	-0.21
Zhwiki	4	1	.79	5.25	2.18	.62	5.34	2.19	0.26	-0.01	-0.02
Zhwiki	4	2	.48	5.10	.98	.40	6.47	.95	0.12	-0.20	0.04
Zhwiki	4	3	.39	4.17	.79	.33	5.22	.90	0.09	-0.15	-0.16
Zhwiki	4	4	.36	2.80	.43	.32	4.45	.53	0.07	-0.24	-0.15
Zhwiki	16	1	.61	5.85	2.61	.57	5.82	2.57	0.06	0.00	0.06
Zhwiki	16	2	.38	3.45	1.16	.34	4.16	1.19	0.06	-0.10	-0.04
Zhwiki	16	3	.31	2.43	.91	.27	3.29	.98	0.06	-0.12	-0.11
Zhwiki	16	4	.29	1.71	.53	.25	2.47	.67	0.06	-0.11	-0.20
Zhwiki	32	1	.58	6.02	2.74	.56	6.02	2.72	0.03	-0.00	0.03
Zhwiki	32	2	.35	3.26	1.20	.34	3.92	1.25	0.02	-0.10	-0.07
Zhwiki	32	3	.28	2.22	.95	.26	3.12	1.06	0.03	-0.13	-0.16
Zhwiki	32	4	.27	1.57	.57	.26	2.22	.72	0.02	-0.10	-0.23
									μ	**0.07**	-0.11	-0.12
									σ	0.06	0.07	0.10

Table D.78: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=@, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.08	40.30	23.40	.78	39.40	22.40	0.21	0.06	0.65
Arwiki	4	2	.67	52.20	11.70	.58	54.80	11.30	0.07	-0.18	0.25
Arwiki	4	3	.59	40.20	7.99	.52	43.50	7.60	0.05	-0.23	0.27
Arwiki	4	4	.56	32.10	5.98	.53	34.10	5.71	0.03	-0.14	0.19
Arwiki	16	1	.82	40.10	23.10	.71	39.80	22.90	0.07	0.02	0.17
Arwiki	16	2	.47	28.70	11.70	.44	32.50	11.40	0.03	-0.27	0.22
Arwiki	16	3	.43	21.50	8.03	.42	24.60	7.65	0.01	-0.21	0.26
Arwiki	16	4	.42	18.90	6.04	.41	17.80	5.75	0.01	0.08	0.20
Arwiki	32	1	.76	39.30	23.10	.70	38.50	23.00	0.04	0.06	0.07
Arwiki	32	2	.45	26.40	11.80	.41	28.20	11.50	0.03	-0.12	0.25
Arwiki	32	3	.40	18.70	8.06	.40	19.20	7.68	0.00	-0.03	0.26
Arwiki	32	4	.40	14.50	6.12	.40	14.50	5.77	0.00	-0.00	0.24
Enwiki	4	1	.74	27.80	16.10	.53	27.20	15.50	0.21	0.06	0.66
Enwiki	4	2	.47	36.40	8.17	.39	37.90	7.83	0.08	-0.15	0.34
Enwiki	4	3	.40	30.60	5.45	.36	29.90	5.25	0.04	0.07	0.20
Enwiki	4	4	.38	23.30	4.11	.35	23.50	3.94	0.03	-0.02	0.17
Enwiki	16	1	.56	27.90	16.00	.49	27.50	15.80	0.07	0.04	0.12
Enwiki	16	2	.33	20.00	8.05	.30	22.40	7.92	0.03	-0.24	0.12
Enwiki	16	3	.29	15.10	5.50	.29	17.10	5.31	0.01	-0.20	0.20
Enwiki	16	4	.29	13.00	4.15	.28	12.30	3.97	0.00	0.07	0.19
Enwiki	32	1	.52	27.20	16.00	.49	26.60	15.90	0.03	0.06	0.04
Enwiki	32	2	.31	18.70	8.12	.28	19.40	7.96	0.02	-0.07	0.16
Enwiki	32	3	.28	12.90	5.54	.27	13.40	5.34	0.01	-0.05	0.21
Enwiki	32	4	.27	10.10	4.21	.28	10.00	3.99	-0.01	0.01	0.23
Ruwiki	4	1	1.34	49.70	28.80	.96	48.60	27.60	0.21	0.06	0.67
Ruwiki	4	2	.82	66.30	14.40	.70	67.50	14.00	0.07	-0.07	0.26
Ruwiki	4	3	.72	53.10	9.72	.65	53.50	9.38	0.04	-0.02	0.19
Ruwiki	4	4	.71	40.60	7.35	.63	41.90	7.04	0.04	-0.08	0.18
Ruwiki	16	1	1.01	49.50	28.40	.89	48.90	28.10	0.07	0.03	0.17
Ruwiki	16	2	.60	35.80	14.40	.56	39.90	14.10	0.03	-0.24	0.16
Ruwiki	16	3	.52	27.20	9.73	.52	30.50	9.44	0.00	-0.18	0.17
Ruwiki	16	4	.52	24.00	7.28	.51	22.00	7.08	0.00	0.11	0.11
Ruwiki	32	1	.94	48.30	28.40	.88	47.60	28.30	0.04	0.04	0.04
Ruwiki	32	2	.53	33.40	14.40	.51	34.80	14.20	0.01	-0.08	0.13
Ruwiki	32	3	.49	23.10	9.85	.49	23.60	9.48	0.00	-0.03	0.21
Ruwiki	32	4	.49	18.00	7.43	.50	18.00	7.12	-0.01	0.00	0.17
Zhwiki	4	1	.51	19.40	11.10	.37	18.90	10.70	0.21	0.07	0.68
Zhwiki	4	2	.33	24.80	5.61	.28	25.90	5.40	0.07	-0.16	0.31
Zhwiki	4	3	.28	19.80	3.80	.25	20.50	3.63	0.05	-0.10	0.25
Zhwiki	4	4	.27	15.20	2.89	.25	16.20	2.73	0.03	-0.15	0.24
Zhwiki	16	1	.39	19.30	11.00	.34	19.10	10.90	0.07	0.03	0.10
Zhwiki	16	2	.23	13.90	5.62	.21	15.50	5.46	0.03	-0.24	0.24
Zhwiki	16	3	.21	10.40	3.84	.20	11.80	3.66	0.01	-0.20	0.26
Zhwiki	16	4	.20	8.65	2.93	.20	8.59	2.74	0.00	0.01	0.27
Zhwiki	32	1	.36	18.80	11.00	.34	18.50	11.00	0.04	0.05	0.07
Zhwiki	32	2	.22	12.90	5.62	.20	13.40	5.48	0.02	-0.08	0.21
Zhwiki	32	3	.20	9.04	3.90	.19	9.08	3.67	0.01	-0.01	0.34
Zhwiki	32	4	.19	7.13	2.91	.20	6.95	2.76	-0.01	0.03	0.22
								μ	**0.04**	-0.05	0.23
								σ	0.06	0.11	0.15

Table D.79: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=@, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	.85	22.80	22.40	.64	22.80	22.40	0.15	0.00	0.00
Arwiki	4	2	.75	28.10	11.30	.73	23.70	11.20	0.01	0.31	0.02
Arwiki	4	3	.84	21.00	7.49	.76	16.90	7.48	0.06	0.29	0.01
Arwiki	4	4	.89	16.50	5.65	.76	12.80	5.61	0.09	0.25	0.03
Arwiki	16	1	.67	22.80	22.40	.59	22.80	22.40	0.06	-0.00	-0.00
Arwiki	16	2	.48	16.00	11.20	.56	17.00	11.20	-0.06	-0.07	0.01
Arwiki	16	3	.53	11.40	7.50	.60	11.60	7.49	-0.05	-0.01	0.01
Arwiki	16	4	.53	8.90	5.66	.61	8.45	5.62	-0.05	0.03	0.03
Arwiki	32	1	.63	22.80	22.40	.58	22.80	22.40	0.03	-0.00	-0.00
Arwiki	32	2	.45	15.00	11.30	.53	15.00	11.20	-0.06	-0.00	0.02
Arwiki	32	3	.49	10.30	7.52	.60	10.30	7.49	-0.08	-0.00	0.02
Arwiki	32	4	.50	7.84	5.66	.61	7.48	5.62	-0.08	0.02	0.03
Enwiki	4	1	.59	15.80	15.50	.44	15.80	15.50	0.15	0.00	-0.00
Enwiki	4	2	.54	19.40	7.76	.51	16.40	7.76	0.03	0.30	-0.01
Enwiki	4	3	.58	14.50	5.19	.52	11.70	5.17	0.06	0.29	0.02
Enwiki	4	4	.60	11.20	3.90	.52	8.84	3.89	0.09	0.24	0.01
Enwiki	16	1	.46	15.70	15.50	.41	15.80	15.50	0.05	-0.01	-0.00
Enwiki	16	2	.33	11.10	7.78	.38	11.70	7.76	-0.05	-0.07	0.01
Enwiki	16	3	.36	7.84	5.18	.42	7.96	5.18	-0.06	-0.01	0.00
Enwiki	16	4	.37	6.16	3.93	.42	5.82	3.88	-0.05	0.03	0.04
Enwiki	32	1	.43	15.80	15.50	.40	15.80	15.50	0.03	-0.00	-0.00
Enwiki	32	2	.31	10.40	7.79	.38	10.40	7.76	-0.07	0.00	0.02
Enwiki	32	3	.34	7.08	5.19	.42	7.26	5.18	-0.08	-0.02	0.01
Enwiki	32	4	.34	5.44	3.93	.42	5.19	3.88	-0.08	0.02	0.04
Ruwiki	4	1	1.06	28.10	27.60	.79	28.10	27.60	0.15	0.00	0.00
Ruwiki	4	2	.95	34.70	13.80	.90	29.20	13.80	0.03	0.31	0.00
Ruwiki	4	3	1.04	25.90	9.26	.93	20.90	9.23	0.06	0.28	0.02
Ruwiki	4	4	1.10	20.10	6.98	.92	15.80	6.94	0.10	0.24	0.02
Ruwiki	16	1	.83	28.10	27.60	.73	28.20	27.60	0.06	-0.01	0.00
Ruwiki	16	2	.59	19.70	13.90	.68	20.90	13.80	-0.05	-0.07	0.01
Ruwiki	16	3	.66	14.00	9.26	.75	14.30	9.23	-0.05	-0.02	0.01
Ruwiki	16	4	.66	10.90	6.96	.74	10.40	6.94	-0.04	0.03	0.02
Ruwiki	32	1	.78	28.10	27.60	.72	28.10	27.60	0.03	-0.00	0.00
Ruwiki	32	2	.55	18.50	13.90	.66	18.50	13.80	-0.06	-0.00	0.02
Ruwiki	32	3	.61	12.60	9.26	.74	12.80	9.24	-0.07	-0.01	0.01
Ruwiki	32	4	.62	9.61	6.97	.75	9.30	6.94	-0.07	0.02	0.02
Zhwiki	4	1	.41	10.90	10.70	.31	10.90	10.70	0.15	0.00	-0.00
Zhwiki	4	2	.37	13.40	5.37	.35	11.30	5.36	0.02	0.30	0.01
Zhwiki	4	3	.40	9.92	3.58	.36	8.03	3.57	0.06	0.28	0.02
Zhwiki	4	4	.42	7.79	2.72	.36	6.14	2.70	0.09	0.24	0.03
Zhwiki	16	1	.32	10.90	10.70	.28	10.90	10.70	0.06	-0.00	0.00
Zhwiki	16	2	.22	7.63	5.36	.27	8.11	5.36	-0.07	-0.07	0.01
Zhwiki	16	3	.25	5.45	3.59	.29	5.49	3.58	-0.05	-0.00	0.02
Zhwiki	16	4	.25	4.25	2.73	.29	4.09	2.68	-0.05	0.02	0.06
Zhwiki	32	1	.31	10.90	10.70	.28	10.90	10.70	0.04	-0.00	0.02
Zhwiki	32	2	.20	7.13	5.36	.26	7.17	5.35	-0.07	-0.01	0.02
Zhwiki	32	3	.23	4.90	3.59	.29	4.91	3.57	-0.08	-0.00	0.02
Zhwiki	32	4	.24	3.80	2.73	.29	3.60	2.69	-0.08	0.03	0.06
								μ	**0.00**	0.07	0.02
								σ	0.07	0.12	0.01

Table D.80: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=@, Colours=never

Arguments: Date -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	2.14	6.38	3.19	1.78	6.35	3.31	0.25	0.00	-0.08
Arwiki	4	2	1.36	7.47	1.25	1.17	9.88	1.18	0.13	-0.17	0.05
Arwiki	4	3	1.30	6.51	1.17	1.13	7.75	1.34	0.12	-0.09	-0.12
Arwiki	4	4	1.30	5.33	.68	1.13	5.79	.65	0.12	-0.03	0.02
Arwiki	16	1	1.75	6.12	3.04	1.69	5.97	3.01	0.04	0.01	0.02
Arwiki	16	2	1.11	4.47	1.40	1.00	5.65	1.62	0.07	-0.08	-0.15
Arwiki	16	3	1.08	3.90	1.41	1.02	4.40	1.47	0.05	-0.03	-0.04
Arwiki	16	4	1.19	2.45	.76	1.01	3.05	.84	0.13	-0.04	-0.06
Arwiki	32	1	1.69	6.00	3.09	1.68	6.10	3.15	0.01	-0.01	-0.05
Arwiki	32	2	1.05	3.88	1.51	1.00	4.57	1.51	0.03	-0.05	0.00
Arwiki	32	3	1.13	2.94	1.33	1.01	3.79	1.50	0.09	-0.06	-0.12
Arwiki	32	4	1.10	2.23	.86	1.03	2.38	.82	0.05	-0.01	0.03
Enwiki	4	1	1.42	3.31	1.58	1.17	3.21	1.72	0.25	0.01	-0.14
Enwiki	4	2	.84	4.87	.42	.72	6.17	.48	0.12	-0.13	-0.06
Enwiki	4	3	.82	4.03	.52	.72	4.53	.57	0.10	-0.05	-0.05
Enwiki	4	4	.82	3.05	.20	.68	3.52	.25	0.14	-0.05	-0.05
Enwiki	16	1	1.16	2.91	1.42	1.11	2.86	1.48	0.04	0.00	-0.06
Enwiki	16	2	.73	1.81	.52	.64	2.80	.54	0.09	-0.10	-0.01
Enwiki	16	3	.74	1.78	.61	.65	2.27	.62	0.08	-0.05	-0.02
Enwiki	16	4	.72	1.21	.26	.62	1.49	.25	0.10	-0.03	0.01
Enwiki	32	1	1.11	2.86	1.45	1.10	2.88	1.53	0.01	-0.00	-0.07
Enwiki	32	2	.68	1.63	.54	.63	2.35	.57	0.05	-0.07	-0.03
Enwiki	32	3	.68	1.59	.67	.65	1.85	.65	0.03	-0.03	0.02
Enwiki	32	4	.70	.98	.31	.63	1.19	.30	0.07	-0.02	0.01
Ruwiki	4	1	2.65	7.93	3.94	2.21	7.92	4.07	0.25	0.00	-0.07
Ruwiki	4	2	1.73	9.13	1.55	1.42	12.80	1.74	0.17	-0.21	-0.11
Ruwiki	4	3	1.58	8.55	1.45	1.41	9.25	1.40	0.10	-0.04	0.03
Ruwiki	4	4	1.60	6.48	.86	1.37	7.61	1.06	0.13	-0.06	-0.11
Ruwiki	16	1	2.17	7.51	3.71	2.10	7.33	3.66	0.04	0.01	0.03
Ruwiki	16	2	1.45	4.96	1.76	1.26	6.77	1.85	0.11	-0.10	-0.05
Ruwiki	16	3	1.39	4.45	1.65	1.27	5.11	1.57	0.07	-0.04	0.05
Ruwiki	16	4	1.38	3.31	1.04	1.20	4.05	1.17	0.10	-0.04	-0.07
Ruwiki	32	1	2.09	7.54	3.87	2.07	7.34	3.74	0.01	0.01	0.07
Ruwiki	32	2	1.35	4.68	1.85	1.21	6.22	2.11	0.08	-0.09	-0.15
Ruwiki	32	3	1.34	3.85	1.69	1.24	4.43	1.67	0.06	-0.03	0.01
Ruwiki	32	4	1.39	2.70	1.06	1.24	3.30	1.16	0.08	-0.03	-0.06
Zhwiki	4	1	1.03	3.13	1.57	.86	3.06	1.59	0.26	0.01	-0.02
Zhwiki	4	2	.66	3.98	.66	.56	4.89	.69	0.15	-0.13	-0.04
Zhwiki	4	3	.62	3.56	.61	.55	3.77	.65	0.10	-0.03	-0.06
Zhwiki	4	4	.61	2.78	.40	.58	2.77	.39	0.05	0.00	0.02
Zhwiki	16	1	.85	2.94	1.49	.82	2.81	1.44	0.05	0.02	0.08
Zhwiki	16	2	.55	2.18	.78	.51	2.55	.71	0.05	-0.05	0.09
Zhwiki	16	3	.57	1.71	.67	.49	2.13	.69	0.11	-0.06	-0.03
Zhwiki	16	4	.55	1.38	.44	.52	1.44	.43	0.04	-0.01	0.02
Zhwiki	32	1	.82	2.96	1.55	.81	2.81	1.48	0.02	0.02	0.11
Zhwiki	32	2	.51	1.97	.79	.49	2.14	.72	0.03	-0.02	0.10
Zhwiki	32	3	.54	1.57	.70	.48	1.97	.77	0.09	-0.06	-0.09
Zhwiki	32	4	.53	1.25	.51	.50	1.28	.47	0.03	-0.01	0.06
								μ	**0.09**	-0.04	-0.02
								σ	0.06	0.05	0.07

Table D.81: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=Date, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.36	49.00	32.80	1.03	47.20	32.10	0.23	0.12	0.54
Arwiki	4	2	.90	57.10	16.80	.82	49.80	16.20	0.06	0.51	0.47
Arwiki	4	3	.91	45.50	11.30	.80	39.10	11.30	0.08	0.45	0.00
Arwiki	4	4	.93	35.30	8.42	.82	30.30	8.12	0.07	0.35	0.21
Arwiki	16	1	1.10	49.30	32.70	.99	48.80	32.50	0.07	0.04	0.14
Arwiki	16	2	.73	33.10	17.10	.68	36.50	17.10	0.03	-0.23	0.02
Arwiki	16	3	.73	25.20	11.50	.71	26.50	11.70	0.02	-0.09	-0.09
Arwiki	16	4	.75	20.50	8.65	.72	19.60	8.54	0.02	0.06	0.08
Arwiki	32	1	1.04	48.10	32.60	.98	47.50	32.50	0.04	0.04	0.04
Arwiki	32	2	.68	31.80	16.90	.69	31.90	16.50	-0.01	-0.01	0.29
Arwiki	32	3	.70	22.60	11.60	.68	23.00	11.70	0.01	-0.03	-0.02
Arwiki	32	4	.73	17.10	8.64	.71	16.50	8.38	0.01	0.04	0.18
Enwiki	4	1	.87	30.60	21.60	.65	29.90	21.10	0.23	0.07	0.50
Enwiki	4	2	.56	33.00	10.90	.47	33.70	10.70	0.08	-0.07	0.24
Enwiki	4	3	.55	29.30	7.13	.47	25.00	6.97	0.07	0.43	0.17
Enwiki	4	4	.57	22.10	5.21	.48	19.60	5.14	0.10	0.26	0.08
Enwiki	16	1	.69	31.50	21.50	.62	30.80	21.20	0.07	0.07	0.28
Enwiki	16	2	.42	21.10	11.00	.41	23.00	10.80	0.01	-0.18	0.22
Enwiki	16	3	.45	15.70	7.36	.42	16.10	7.19	0.03	-0.03	0.18
Enwiki	16	4	.44	12.90	5.35	.42	12.20	5.28	0.02	0.07	0.08
Enwiki	32	1	.65	30.40	21.50	.61	29.90	21.40	0.04	0.05	0.10
Enwiki	32	2	.41	19.90	11.00	.40	20.40	10.80	0.01	-0.05	0.17
Enwiki	32	3	.42	14.20	7.35	.41	13.90	7.20	0.01	0.02	0.15
Enwiki	32	4	.43	10.50	5.45	.41	10.30	5.34	0.02	0.02	0.11
Ruwiki	4	1	1.69	60.10	40.30	1.28	58.30	39.40	0.23	0.10	0.56
Ruwiki	4	2	1.22	58.10	20.90	.98	62.60	20.70	0.13	-0.25	0.11
Ruwiki	4	3	1.16	49.10	14.10	1.00	47.50	13.40	0.09	0.09	0.39
Ruwiki	4	4	1.14	39.80	10.70	.98	38.90	10.70	0.09	0.05	-0.02
Ruwiki	16	1	1.35	60.80	40.10	1.22	59.80	39.60	0.07	0.06	0.28
Ruwiki	16	2	.89	41.00	21.00	.85	44.50	20.70	0.02	-0.20	0.17
Ruwiki	16	3	.90	31.00	14.40	.88	31.80	13.80	0.01	-0.05	0.37
Ruwiki	16	4	.91	24.70	10.80	.86	25.00	10.90	0.03	-0.02	-0.08
Ruwiki	32	1	1.28	59.60	40.10	1.21	58.30	39.90	0.04	0.07	0.09
Ruwiki	32	2	.86	38.50	20.90	.79	41.00	21.10	0.04	-0.14	-0.07
Ruwiki	32	3	.88	27.60	14.40	.84	28.00	14.00	0.02	-0.02	0.17
Ruwiki	32	4	.88	21.50	10.90	.85	21.40	10.70	0.02	0.00	0.13
Zhwiki	4	1	.67	24.00	15.70	.51	23.10	15.40	0.23	0.13	0.45
Zhwiki	4	2	.44	27.20	8.12	.40	24.40	8.07	0.07	0.41	0.06
Zhwiki	4	3	.45	21.60	5.54	.39	19.20	5.57	0.09	0.35	-0.04
Zhwiki	4	4	.46	15.80	4.23	.41	15.00	4.03	0.08	0.12	0.29
Zhwiki	16	1	.54	24.10	15.60	.48	23.70	15.50	0.07	0.06	0.15
Zhwiki	16	2	.35	16.60	8.17	.34	17.40	7.96	0.01	-0.12	0.31
Zhwiki	16	3	.36	12.40	5.65	.35	13.00	5.54	0.02	-0.08	0.15
Zhwiki	16	4	.36	9.85	4.32	.36	9.56	4.11	0.00	0.04	0.31
Zhwiki	32	1	.51	23.50	15.60	.48	22.90	15.50	0.04	0.08	0.21
Zhwiki	32	2	.34	15.50	8.15	.34	15.50	7.88	0.00	-0.00	0.39
Zhwiki	32	3	.35	11.00	5.67	.33	11.60	5.79	0.03	-0.09	-0.18
Zhwiki	32	4	.35	8.60	4.30	.35	8.30	4.15	-0.00	0.04	0.21
								μ	**0.06**	0.05	0.18
								σ	0.06	0.17	0.17

Table D.82: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=Date, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.03	24.60	23.70	.80	24.70	23.70	0.16	-0.00	0.00
Arwiki	4	2	.94	30.00	12.00	.88	25.90	11.90	0.04	0.28	0.06
Arwiki	4	3	.99	23.20	8.35	.87	19.90	8.55	0.09	0.23	-0.14
Arwiki	4	4	1.04	18.40	6.37	.92	15.20	6.15	0.08	0.22	0.15
Arwiki	16	1	.82	24.60	23.70	.73	24.70	23.70	0.06	-0.01	-0.00
Arwiki	16	2	.61	18.30	12.50	.63	19.90	12.50	-0.01	-0.12	0.02
Arwiki	16	3	.69	13.10	8.33	.67	14.00	8.56	0.01	-0.06	-0.16
Arwiki	16	4	.71	10.40	6.42	.70	10.30	6.37	0.01	0.01	0.03
Arwiki	32	1	.78	24.60	23.70	.73	24.60	23.70	0.03	-0.00	-0.00
Arwiki	32	2	.61	16.50	12.00	.64	16.40	12.00	-0.02	0.01	0.01
Arwiki	32	3	.65	11.60	8.27	.65	12.20	8.56	0.00	-0.04	-0.20
Arwiki	32	4	.67	8.97	6.35	.69	8.38	6.16	-0.02	0.04	0.13
Enwiki	4	1	.64	15.80	15.50	.48	15.80	15.50	0.17	-0.00	0.00
Enwiki	4	2	.52	19.80	7.77	.48	17.10	7.76	0.04	0.28	0.01
Enwiki	4	3	.57	14.90	5.18	.49	12.40	5.18	0.08	0.26	0.01
Enwiki	4	4	.59	11.70	3.92	.49	9.48	3.88	0.10	0.22	0.03
Enwiki	16	1	.49	15.70	15.50	.43	15.80	15.50	0.06	-0.01	-0.00
Enwiki	16	2	.36	11.30	7.79	.36	12.60	7.75	-0.00	-0.13	0.04
Enwiki	16	3	.38	8.23	5.19	.38	8.62	5.17	0.00	-0.04	0.02
Enwiki	16	4	.39	6.53	3.93	.39	6.37	3.88	0.00	0.02	0.05
Enwiki	32	1	.46	15.70	15.50	.43	15.70	15.50	0.03	-0.00	-0.00
Enwiki	32	2	.34	10.70	7.78	.34	10.60	7.76	-0.01	0.00	0.02
Enwiki	32	3	.35	7.32	5.20	.36	7.27	5.17	-0.00	0.01	0.03
Enwiki	32	4	.36	5.62	3.92	.36	5.28	3.88	-0.00	0.04	0.04
Ruwiki	4	1	1.27	30.30	29.20	.98	30.30	29.20	0.16	-0.00	-0.01
Ruwiki	4	2	1.12	37.50	15.20	1.01	32.70	15.20	0.06	0.27	-0.04
Ruwiki	4	3	1.23	28.40	10.10	1.08	24.20	10.10	0.08	0.24	0.01
Ruwiki	4	4	1.25	22.80	7.92	1.06	19.60	8.18	0.11	0.18	-0.15
Ruwiki	16	1	1.01	30.20	29.20	.90	30.30	29.20	0.06	-0.01	-0.01
Ruwiki	16	2	.80	22.10	15.10	.79	24.20	15.10	0.01	-0.12	0.01
Ruwiki	16	3	.83	16.30	10.40	.84	16.80	10.00	-0.01	-0.03	0.17
Ruwiki	16	4	.86	12.80	7.93	.81	13.10	8.18	0.03	-0.01	-0.14
Ruwiki	32	1	.95	30.20	29.20	.90	30.20	29.20	0.03	-0.00	-0.01
Ruwiki	32	2	.70	21.10	15.40	.71	21.00	15.40	-0.00	0.01	0.01
Ruwiki	32	3	.79	14.40	10.20	.79	14.80	10.30	0.00	-0.02	-0.07
Ruwiki	32	4	.80	11.20	7.99	.81	10.90	7.92	-0.01	0.02	0.04
Zhwiki	4	1	.51	11.90	11.30	.39	11.90	11.30	0.16	-0.00	0.01
Zhwiki	4	2	.46	14.70	5.93	.42	12.80	5.93	0.06	0.28	-0.01
Zhwiki	4	3	.48	11.30	4.09	.42	9.79	4.16	0.09	0.23	-0.11
Zhwiki	4	4	.49	9.02	3.15	.45	7.60	3.05	0.07	0.21	0.15
Zhwiki	16	1	.40	11.90	11.30	.36	11.90	11.30	0.06	-0.01	0.01
Zhwiki	16	2	.33	8.71	5.82	.33	9.42	5.81	-0.01	-0.10	0.02
Zhwiki	16	3	.33	6.54	4.11	.33	6.82	4.08	0.01	-0.04	0.05
Zhwiki	16	4	.34	5.21	3.18	.35	5.03	3.04	-0.01	0.03	0.20
Zhwiki	32	1	.38	11.90	11.30	.36	11.90	11.30	0.03	-0.00	0.01
Zhwiki	32	2	.31	8.00	5.72	.32	7.93	5.71	-0.02	0.01	0.02
Zhwiki	32	3	.32	5.70	4.01	.30	6.23	4.27	0.03	-0.08	-0.38
Zhwiki	32	4	.32	4.55	3.21	.34	4.22	3.05	-0.03	0.05	0.22
								μ	**0.04**	0.05	0.00
								σ	0.05	0.12	0.10

Table D.83: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=Date, Colours=never

Arguments: Email -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	4.18	1.71	1.28	3.75	7.83	3.61	0.30	-0.43	-1.63
Arwiki	4	2	2.38	6.14	.42	2.12	12.30	1.45	0.18	-0.43	-0.72
Arwiki	4	3	2.43	5.54	.59	2.37	25.70	1.41	0.04	-1.41	-0.57
Arwiki	4	4	2.48	4.44	.23	2.58	28.00	.74	-0.07	-1.65	-0.36
Arwiki	16	1	3.68	1.59	1.26	3.62	8.18	3.84	0.04	-0.46	-1.80
Arwiki	16	2	2.17	2.13	.51	2.02	9.57	1.74	0.11	-0.52	-0.86
Arwiki	16	3	2.19	2.30	.80	2.28	23.70	1.53	-0.06	-1.49	-0.51
Arwiki	16	4	2.26	1.41	.24	2.46	25.40	.85	-0.14	-1.67	-0.43
Arwiki	32	1	3.59	1.65	1.33	3.60	8.39	4.06	-0.01	-0.47	-1.91
Arwiki	32	2	2.11	1.59	.52	1.98	8.89	1.75	0.09	-0.51	-0.86
Arwiki	32	3	2.36	1.45	.66	2.26	23.60	1.57	0.07	-1.54	-0.64
Arwiki	32	4	2.25	.89	.22	2.46	25.30	.89	-0.15	-1.71	-0.46
Enwiki	4	1	2.14	6.03	2.65	1.80	5.23	2.40	0.34	0.08	0.25
Enwiki	4	2	1.23	6.49	1.04	1.04	7.22	.95	0.20	-0.07	0.09
Enwiki	4	3	.99	5.33	.89	.82	6.42	.92	0.17	-0.11	-0.03
Enwiki	4	4	.96	3.82	.45	.78	5.87	.49	0.18	-0.21	-0.04
Enwiki	16	1	1.80	6.10	2.80	1.71	5.42	2.52	0.09	0.07	0.29
Enwiki	16	2	1.04	3.95	1.23	.94	4.62	1.15	0.10	-0.07	0.07
Enwiki	16	3	.80	3.02	1.12	.72	4.41	1.04	0.08	-0.14	0.08
Enwiki	16	4	.74	2.11	.56	.65	3.96	.55	0.10	-0.19	0.01
Enwiki	32	1	1.74	6.15	2.87	1.70	5.50	2.60	0.04	0.07	0.27
Enwiki	32	2	1.00	3.46	1.25	.93	4.30	1.18	0.07	-0.08	0.07
Enwiki	32	3	.75	2.58	1.09	.65	3.93	1.11	0.09	-0.14	-0.01
Enwiki	32	4	.74	1.69	.55	.62	3.38	.60	0.12	-0.17	-0.05
Ruwiki	4	1	5.46	1.95	1.50	4.94	9.60	4.34	0.30	-0.43	-1.61
Ruwiki	4	2	3.09	8.57	.49	2.87	17.70	1.85	0.12	-0.52	-0.77
Ruwiki	4	3	3.35	6.39	.79	3.38	37.60	1.69	-0.01	-1.77	-0.51
Ruwiki	4	4	3.39	5.40	.26	3.74	40.90	.88	-0.20	-2.01	-0.35
Ruwiki	16	1	4.83	1.84	1.47	4.76	10.10	4.64	0.04	-0.47	-1.79
Ruwiki	16	2	2.92	2.66	.58	2.70	13.60	2.16	0.13	-0.62	-0.89
Ruwiki	16	3	3.05	2.84	.94	3.30	35.40	1.77	-0.14	-1.84	-0.47
Ruwiki	16	4	3.12	1.76	.28	3.64	36.60	1.00	-0.30	-1.97	-0.41
Ruwiki	32	1	4.71	1.91	1.55	4.73	10.30	4.91	-0.01	-0.48	-1.91
Ruwiki	32	2	2.94	1.76	.56	2.59	11.20	2.16	0.20	-0.54	-0.91
Ruwiki	32	3	3.10	1.87	.88	3.28	35.40	1.86	-0.10	-1.90	-0.55
Ruwiki	32	4	3.08	1.18	.32	3.57	37.50	1.10	-0.28	-2.06	-0.45
Zhwiki	4	1	2.40	.90	.62	2.23	3.68	1.67	0.24	-0.41	-1.54
Zhwiki	4	2	1.58	2.45	.22	1.48	11.40	.72	0.15	-1.31	-0.74
Zhwiki	4	3	1.60	2.68	.28	1.67	19.10	.68	-0.10	-2.41	-0.59
Zhwiki	4	4	1.65	1.98	.12	1.80	20.10	.36	-0.23	-2.67	-0.36
Zhwiki	16	1	2.16	.88	.61	2.16	3.89	1.80	-0.01	-0.44	-1.74
Zhwiki	16	2	1.36	1.28	.25	1.36	8.57	.88	-0.01	-1.07	-0.92
Zhwiki	16	3	1.51	1.07	.35	1.63	18.20	.70	-0.19	-2.51	-0.52
Zhwiki	16	4	1.52	.73	.12	1.76	19.00	.39	-0.36	-2.69	-0.39
Zhwiki	32	1	2.12	.91	.65	2.16	3.95	1.87	-0.07	-0.45	-1.79
Zhwiki	32	2	1.41	.82	.25	1.35	8.15	.83	0.09	-1.08	-0.86
Zhwiki	32	3	1.49	.77	.33	1.62	17.90	.72	-0.20	-2.52	-0.57
Zhwiki	32	4	1.59	.45	.13	1.74	18.80	.44	-0.22	-2.69	-0.45
								μ	**0.02**	-1.00	-0.64
								σ	0.16	0.88	0.62

Table D.84: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=Email, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	2.42	48.60	33.80	2.04	40.20	24.90	0.27	0.58	6.18
Arwiki	4	2	1.37	59.10	17.00	1.24	55.70	12.30	0.09	0.24	3.28
Arwiki	4	3	1.37	44.70	11.50	1.56	76.30	8.09	-0.13	-2.20	2.39
Arwiki	4	4	1.43	34.60	8.61	1.67	68.20	6.10	-0.17	-2.35	1.75
Arwiki	16	1	2.06	49.00	33.40	1.97	42.40	25.10	0.07	0.47	5.78
Arwiki	16	2	1.15	34.90	16.90	1.13	40.60	12.30	0.02	-0.40	3.21
Arwiki	16	3	1.17	25.20	11.50	1.38	60.30	8.25	-0.15	-2.45	2.24
Arwiki	16	4	1.23	19.60	8.57	1.51	55.60	6.18	-0.19	-2.51	1.67
Arwiki	32	1	1.99	47.10	33.40	1.96	41.30	25.20	0.02	0.41	5.73
Arwiki	32	2	1.14	30.50	17.00	1.11	36.40	12.30	0.02	-0.41	3.23
Arwiki	32	3	1.17	21.10	11.40	1.35	56.50	8.26	-0.12	-2.47	2.20
Arwiki	32	4	1.19	16.20	8.55	1.45	51.20	6.19	-0.18	-2.44	1.65
Enwiki	4	1	1.41	30.40	18.60	1.10	28.10	17.10	0.30	0.24	1.51
Enwiki	4	2	.85	35.20	9.44	.68	38.30	8.56	0.17	-0.31	0.89
Enwiki	4	3	.69	29.80	6.33	.58	34.90	5.56	0.11	-0.52	0.77
Enwiki	4	4	.65	22.60	4.75	.56	30.50	4.17	0.08	-0.79	0.58
Enwiki	16	1	1.14	31.40	18.50	1.06	29.20	17.40	0.09	0.22	1.11
Enwiki	16	2	.67	21.30	9.50	.58	24.80	8.54	0.09	-0.36	0.97
Enwiki	16	3	.53	16.20	6.25	.46	21.90	5.72	0.07	-0.57	0.54
Enwiki	16	4	.53	13.70	4.74	.42	19.90	4.24	0.11	-0.62	0.50
Enwiki	32	1	1.09	29.90	18.50	1.05	28.40	17.40	0.04	0.16	1.08
Enwiki	32	2	.58	20.10	9.23	.58	21.90	8.51	0.00	-0.18	0.72
Enwiki	32	3	.53	13.90	6.47	.41	18.40	5.73	0.12	-0.46	0.75
Enwiki	32	4	.51	10.70	4.77	.41	17.80	4.27	0.11	-0.72	0.51
Ruwiki	4	1	3.22	60.30	42.00	2.74	50.20	30.40	0.27	0.57	6.58
Ruwiki	4	2	1.86	71.90	21.20	1.72	73.80	15.10	0.08	-0.11	3.41
Ruwiki	4	3	1.93	53.80	14.30	2.19	102.00	9.99	-0.15	-2.76	2.43
Ruwiki	4	4	1.98	41.30	10.70	2.36	91.90	7.49	-0.21	-2.86	1.82
Ruwiki	16	1	2.76	61.20	41.70	2.65	52.30	30.90	0.06	0.51	6.09
Ruwiki	16	2	1.60	43.80	21.00	1.54	53.20	15.20	0.03	-0.54	3.29
Ruwiki	16	3	1.73	30.40	14.20	2.02	85.30	10.20	-0.16	-3.11	2.28
Ruwiki	16	4	1.73	24.60	10.60	2.18	77.10	7.66	-0.25	-2.97	1.67
Ruwiki	32	1	2.66	58.70	41.60	2.64	51.00	31.00	0.02	0.44	5.99
Ruwiki	32	2	1.57	38.20	21.00	1.47	45.10	15.20	0.06	-0.39	3.29
Ruwiki	32	3	1.68	26.20	14.20	1.97	79.70	10.20	-0.17	-3.03	2.25
Ruwiki	32	4	1.70	20.00	10.60	2.11	72.00	7.68	-0.23	-2.95	1.65
Zhwiki	4	1	1.40	24.10	16.50	1.22	19.60	11.80	0.27	0.65	6.92
Zhwiki	4	2	.90	26.20	8.34	.84	34.50	5.88	0.09	-1.22	3.62
Zhwiki	4	3	.93	20.10	5.58	.95	43.20	3.87	-0.03	-3.39	2.52
Zhwiki	4	4	.91	16.70	4.16	1.01	38.00	2.91	-0.14	-3.13	1.83
Zhwiki	16	1	1.23	24.50	16.40	1.19	20.40	12.00	0.06	0.60	6.44
Zhwiki	16	2	.80	17.10	8.25	.73	24.00	5.87	0.10	-1.00	3.49
Zhwiki	16	3	.82	12.20	5.55	.88	36.00	3.95	-0.08	-3.50	2.35
Zhwiki	16	4	.84	9.88	4.13	.93	32.20	2.97	-0.14	-3.28	1.72
Zhwiki	32	1	1.19	23.40	16.30	1.19	19.80	12.00	0.01	0.53	6.34
Zhwiki	32	2	.79	15.10	8.25	.71	21.40	5.86	0.11	-0.94	3.52
Zhwiki	32	3	.79	10.50	5.52	.86	33.50	3.96	-0.09	-3.37	2.29
Zhwiki	32	4	.84	7.89	4.14	.91	30.20	2.98	-0.11	-3.28	1.70
									μ	**0.00**	-1.17	2.77
									σ	0.14	1.42	1.88

Table D.85: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=Email, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.79	23.00	22.40	1.54	22.90	22.40	0.18	0.00	0.00
Arwiki	4	2	1.20	28.80	11.30	1.13	29.20	11.20	0.05	-0.03	0.04
Arwiki	4	3	1.30	21.90	7.50	1.21	24.00	7.48	0.06	-0.15	0.01
Arwiki	4	4	1.34	17.40	5.64	1.27	19.10	5.62	0.05	-0.12	0.02
Arwiki	16	1	1.49	23.00	22.40	1.48	23.00	22.40	0.01	-0.00	0.00
Arwiki	16	2	.95	17.90	11.20	.91	19.60	11.20	0.03	-0.12	0.02
Arwiki	16	3	.99	12.40	7.50	.89	16.10	7.49	0.07	-0.26	0.01
Arwiki	16	4	1.00	9.90	5.64	1.05	13.50	5.61	-0.04	-0.25	0.02
Arwiki	32	1	1.43	22.90	22.40	1.47	23.00	22.40	-0.03	-0.00	-0.00
Arwiki	32	2	.90	15.30	11.30	.87	17.00	11.20	0.02	-0.12	0.02
Arwiki	32	3	.95	10.60	7.50	.85	14.10	7.48	0.07	-0.25	0.02
Arwiki	32	4	.96	7.99	5.65	1.01	11.90	5.62	-0.03	-0.28	0.02
Enwiki	4	1	1.12	15.90	15.50	.91	15.90	15.50	0.21	0.00	0.00
Enwiki	4	2	.79	21.10	7.76	.69	20.20	7.77	0.10	0.09	-0.01
Enwiki	4	3	.73	16.40	5.19	.64	16.10	5.17	0.10	0.03	0.01
Enwiki	4	4	.75	13.10	3.92	.62	12.50	3.88	0.13	0.06	0.04
Enwiki	16	1	.90	15.90	15.50	.86	15.90	15.50	0.04	-0.00	0.00
Enwiki	16	2	.56	12.50	7.79	.54	13.20	7.76	0.03	-0.07	0.03
Enwiki	16	3	.55	8.74	5.19	.44	9.77	5.18	0.11	-0.10	0.01
Enwiki	16	4	.55	6.98	3.93	.45	7.75	3.88	0.11	-0.08	0.05
Enwiki	32	1	.85	15.90	15.50	.86	15.90	15.50	-0.00	-0.00	0.00
Enwiki	32	2	.51	10.80	7.80	.50	11.20	7.76	0.02	-0.05	0.04
Enwiki	32	3	.52	7.40	5.20	.39	8.06	5.18	0.14	-0.07	0.02
Enwiki	32	4	.53	5.66	3.93	.41	6.50	3.88	0.12	-0.08	0.04
Ruwiki	4	1	2.43	28.40	27.60	2.11	28.30	27.60	0.18	0.00	-0.00
Ruwiki	4	2	1.71	35.40	13.80	1.54	37.00	13.90	0.10	-0.09	-0.02
Ruwiki	4	3	1.79	26.90	9.25	1.80	31.30	9.23	-0.00	-0.25	0.01
Ruwiki	4	4	1.82	21.60	6.97	1.97	25.30	6.93	-0.08	-0.21	0.02
Ruwiki	16	1	2.05	28.30	27.60	2.03	28.40	27.60	0.01	-0.00	-0.00
Ruwiki	16	2	1.35	22.00	13.90	1.29	25.00	13.80	0.03	-0.17	0.03
Ruwiki	16	3	1.43	15.30	9.25	1.45	22.00	9.23	-0.01	-0.38	0.01
Ruwiki	16	4	1.46	12.10	6.96	1.72	18.30	6.93	-0.15	-0.35	0.01
Ruwiki	32	1	1.97	28.30	27.60	2.02	28.30	27.60	-0.03	-0.00	0.00
Ruwiki	32	2	1.33	19.00	13.90	1.18	21.10	13.80	0.08	-0.12	0.02
Ruwiki	32	3	1.39	13.00	9.26	1.41	19.60	9.24	-0.01	-0.38	0.01
Ruwiki	32	4	1.39	9.85	6.96	1.66	16.40	6.93	-0.15	-0.37	0.01
Zhwiki	4	1	1.04	11.00	10.70	.92	11.00	10.70	0.18	0.00	0.00
Zhwiki	4	2	.76	13.80	5.34	.69	14.80	5.35	0.09	-0.14	-0.02
Zhwiki	4	3	.78	10.20	3.59	.79	12.50	3.59	-0.02	-0.33	0.00
Zhwiki	4	4	.79	8.37	2.69	.87	10.20	2.69	-0.12	-0.27	0.00
Zhwiki	16	1	.90	11.00	10.70	.89	11.00	10.70	0.01	-0.00	0.00
Zhwiki	16	2	.63	8.46	5.36	.58	10.10	5.35	0.07	-0.24	0.02
Zhwiki	16	3	.66	5.94	3.58	.67	9.11	3.58	-0.03	-0.47	-0.00
Zhwiki	16	4	.66	4.62	2.70	.79	7.58	2.68	-0.19	-0.43	0.03
Zhwiki	32	1	.87	11.00	10.70	.89	11.00	10.70	-0.03	-0.00	-0.00
Zhwiki	32	2	.61	7.33	5.37	.53	8.41	5.35	0.12	-0.16	0.03
Zhwiki	32	3	.64	5.05	3.58	.66	8.17	3.57	-0.03	-0.46	0.02
Zhwiki	32	4	.65	3.84	2.70	.77	6.88	2.68	-0.18	-0.45	0.03
								μ	**0.03**	-0.15	0.01
								σ	0.09	0.15	0.01

Table D.86: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=Email, Colours=never

Arguments: URIOrEmail -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	4.46	16.50	7.86	3.79	16.10	7.74	0.46	0.03	0.08
Arwiki	4	2	3.02	18.70	4.33	2.76	16.80	3.78	0.18	0.13	0.39
Arwiki	4	3	2.68	14.00	3.84	2.41	14.40	3.09	0.19	-0.03	0.52
Arwiki	4	4	2.52	11.70	3.09	2.19	13.30	3.05	0.23	-0.11	0.03
Arwiki	16	1	3.86	16.80	7.93	3.66	16.40	7.74	0.14	0.03	0.14
Arwiki	16	2	2.63	12.50	4.91	2.55	10.70	3.75	0.05	0.12	0.81
Arwiki	16	3	2.28	9.66	3.88	1.89	12.70	4.72	0.27	-0.21	-0.58
Arwiki	16	4	2.16	8.00	3.16	1.99	8.16	2.75	0.12	-0.01	0.29
Arwiki	32	1	3.78	17.20	8.17	3.67	16.80	8.15	0.08	0.03	0.01
Arwiki	32	2	2.58	11.90	4.84	2.36	12.10	5.12	0.15	-0.01	-0.19
Arwiki	32	3	2.16	9.68	4.08	1.91	10.20	4.05	0.17	-0.04	0.02
Arwiki	32	4	2.07	8.55	3.67	1.96	7.68	2.98	0.07	0.06	0.48
Enwiki	4	1	2.44	3.40	1.60	2.05	3.27	1.80	0.39	0.01	-0.20
Enwiki	4	2	1.38	5.92	.58	1.14	5.81	.61	0.24	0.01	-0.03
Enwiki	4	3	1.14	5.04	.55	.97	5.62	.66	0.17	-0.06	-0.11
Enwiki	4	4	1.13	3.62	.23	.94	3.99	.28	0.19	-0.04	-0.05
Enwiki	16	1	2.04	3.12	1.50	1.96	3.05	1.48	0.08	0.01	0.02
Enwiki	16	2	1.16	2.46	.53	1.05	2.41	.60	0.11	0.00	-0.07
Enwiki	16	3	.93	2.14	.65	.84	2.93	.57	0.10	-0.08	0.08
Enwiki	16	4	.97	1.37	.21	.84	1.93	.26	0.14	-0.06	-0.05
Enwiki	32	1	1.99	3.05	1.45	1.96	3.08	1.52	0.02	-0.00	-0.07
Enwiki	32	2	1.12	2.08	.54	1.06	2.27	.60	0.06	-0.02	-0.06
Enwiki	32	3	.91	1.63	.57	.82	2.62	.62	0.09	-0.10	-0.05
Enwiki	32	4	.93	1.09	.26	.82	1.74	.30	0.11	-0.07	-0.04
Ruwiki	4	1	5.50	19.20	9.15	4.70	18.70	8.81	0.46	0.03	0.19
Ruwiki	4	2	3.57	23.70	5.66	3.23	21.50	5.05	0.20	0.13	0.34
Ruwiki	4	3	3.19	17.80	4.70	2.77	18.90	4.25	0.24	-0.06	0.26
Ruwiki	4	4	2.97	14.40	3.81	2.73	16.20	3.51	0.14	-0.10	0.17
Ruwiki	16	1	4.76	19.50	9.25	4.53	18.80	8.74	0.13	0.04	0.28
Ruwiki	16	2	3.41	13.90	5.53	2.86	15.80	5.77	0.31	-0.11	-0.14
Ruwiki	16	3	2.74	11.60	4.47	2.39	13.20	4.60	0.19	-0.09	-0.08
Ruwiki	16	4	2.49	10.60	4.12	2.29	10.60	3.58	0.11	-0.00	0.31
Ruwiki	32	1	4.66	20.10	9.53	4.53	19.40	9.18	0.08	0.04	0.20
Ruwiki	32	2	3.28	12.40	4.82	2.98	12.40	4.97	0.17	-0.00	-0.08
Ruwiki	32	3	2.66	9.78	4.04	2.38	10.60	4.04	0.15	-0.05	-0.00
Ruwiki	32	4	2.45	9.59	4.04	2.27	9.45	3.62	0.10	0.01	0.23
Zhwiki	4	1	2.23	8.37	4.00	1.94	8.05	3.84	0.43	0.05	0.24
Zhwiki	4	2	1.52	8.88	2.59	1.22	10.70	3.04	0.44	-0.27	-0.66
Zhwiki	4	3	1.36	7.04	1.81	1.21	7.64	1.69	0.22	-0.09	0.18
Zhwiki	4	4	1.22	6.44	1.77	1.13	7.11	1.50	0.12	-0.10	0.39
Zhwiki	16	1	1.94	8.34	3.99	1.87	8.10	3.81	0.10	0.04	0.25
Zhwiki	16	2	1.37	5.77	2.15	1.15	7.77	2.98	0.33	-0.29	-1.22
Zhwiki	16	3	1.08	5.45	2.21	.86	7.79	3.06	0.32	-0.34	-1.25
Zhwiki	16	4	1.07	4.63	1.84	.93	5.48	1.93	0.20	-0.12	-0.14
Zhwiki	32	1	1.91	8.74	4.20	1.87	8.34	3.98	0.05	0.06	0.31
Zhwiki	32	2	1.30	5.98	2.48	1.21	6.05	2.47	0.13	-0.01	0.01
Zhwiki	32	3	1.07	5.28	2.28	.85	6.84	2.88	0.33	-0.23	-0.88
Zhwiki	32	4	1.07	3.98	1.70	.88	5.32	2.15	0.27	-0.20	-0.66
								μ	**0.19**	-0.04	-0.01
								σ	0.11	0.10	0.40

Table D.87: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=URIOrEmail, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	3.25	77.70	47.40	2.67	74.10	47.50	0.41	0.25	-0.06
Arwiki	4	2	2.35	94.80	26.50	2.20	75.30	23.90	0.11	1.36	1.79
Arwiki	4	3	2.01	81.70	21.20	2.01	64.00	16.90	-0.00	1.24	3.01
Arwiki	4	4	1.95	65.70	16.70	1.86	56.90	15.40	0.06	0.61	0.96
Arwiki	16	1	2.80	79.50	47.60	2.60	76.70	47.60	0.14	0.20	0.03
Arwiki	16	2	1.95	61.50	25.90	1.98	51.10	23.40	-0.02	0.73	1.76
Arwiki	16	3	1.68	46.60	20.30	1.58	49.60	21.80	0.07	-0.21	-1.05
Arwiki	16	4	1.68	38.50	16.90	1.69	34.40	14.10	-0.00	0.29	1.98
Arwiki	32	1	2.71	79.90	47.70	2.59	76.10	47.70	0.08	0.27	-0.00
Arwiki	32	2	1.87	53.70	27.10	1.82	50.90	27.40	0.04	0.20	-0.25
Arwiki	32	3	1.66	40.10	19.40	1.59	40.70	19.30	0.05	-0.05	0.05
Arwiki	32	4	1.60	34.40	16.90	1.63	30.80	14.30	-0.02	0.25	1.80
Enwiki	4	1	1.60	32.30	22.10	1.23	30.80	22.10	0.37	0.15	0.07
Enwiki	4	2	.92	43.40	11.10	.75	37.90	11.00	0.18	0.55	0.08
Enwiki	4	3	.77	33.10	7.33	.63	34.40	7.26	0.14	-0.14	0.06
Enwiki	4	4	.71	27.20	5.44	.59	26.40	5.42	0.12	0.08	0.03
Enwiki	16	1	1.29	32.70	22.20	1.18	31.70	22.20	0.11	0.10	-0.02
Enwiki	16	2	.70	27.50	11.10	.65	22.00	11.10	0.06	0.56	0.05
Enwiki	16	3	.55	18.70	7.40	.51	21.30	7.25	0.04	-0.26	0.15
Enwiki	16	4	.54	15.40	5.51	.51	15.40	5.44	0.03	-0.00	0.07
Enwiki	32	1	1.25	32.50	22.20	1.19	31.10	22.20	0.06	0.14	-0.09
Enwiki	32	2	.70	21.20	11.20	.63	17.70	11.10	0.07	0.35	0.12
Enwiki	32	3	.52	15.80	7.44	.49	18.50	7.24	0.04	-0.27	0.20
Enwiki	32	4	.51	12.10	5.51	.49	13.20	5.45	0.02	-0.11	0.06
Ruwiki	4	1	3.96	94.10	56.50	3.25	88.00	56.50	0.40	0.34	0.02
Ruwiki	4	2	2.77	114.00	32.30	2.54	94.00	30.70	0.13	1.16	0.91
Ruwiki	4	3	2.35	97.90	25.40	2.29	81.80	22.30	0.04	0.92	1.79
Ruwiki	4	4	2.29	78.60	19.40	2.28	66.20	18.00	0.01	0.70	0.81
Ruwiki	16	1	3.40	94.40	56.60	3.16	90.90	56.60	0.14	0.20	0.00
Ruwiki	16	2	2.29	76.20	32.50	2.19	69.00	32.80	0.06	0.41	-0.18
Ruwiki	16	3	1.96	55.90	23.60	1.96	54.70	22.90	-0.00	0.06	0.38
Ruwiki	16	4	1.82	47.90	20.40	1.89	44.10	18.00	-0.04	0.22	1.40
Ruwiki	32	1	3.29	97.80	56.60	3.16	91.40	56.50	0.08	0.36	0.06
Ruwiki	32	2	2.19	67.40	33.20	2.27	57.30	29.00	-0.05	0.57	2.36
Ruwiki	32	3	1.80	49.90	24.80	1.93	45.10	20.90	-0.08	0.27	2.18
Ruwiki	32	4	1.80	38.00	18.80	1.84	38.70	17.80	-0.02	-0.04	0.58
Zhwiki	4	1	1.63	38.10	23.10	1.35	36.50	23.00	0.40	0.24	0.09
Zhwiki	4	2	1.19	42.20	13.50	.98	42.80	15.40	0.31	-0.09	-2.79
Zhwiki	4	3	1.08	33.80	9.39	1.01	32.40	8.68	0.11	0.20	1.04
Zhwiki	4	4	.98	32.00	8.43	.98	28.20	7.24	-0.00	0.55	1.75
Zhwiki	16	1	1.41	38.90	23.10	1.32	37.80	23.10	0.13	0.16	-0.05
Zhwiki	16	2	1.03	29.10	13.40	.89	31.00	15.10	0.22	-0.29	-2.43
Zhwiki	16	3	.83	23.40	10.20	.73	28.00	13.00	0.15	-0.68	-4.10
Zhwiki	16	4	.82	19.50	8.44	.80	20.10	8.63	0.02	-0.09	-0.28
Zhwiki	32	1	1.37	38.80	23.10	1.32	37.40	23.10	0.07	0.21	0.03
Zhwiki	32	2	.99	26.70	13.20	.94	25.30	13.10	0.08	0.20	0.21
Zhwiki	32	3	.82	20.30	10.00	.72	24.40	12.50	0.15	-0.60	-3.64
Zhwiki	32	4	.80	16.40	8.03	.75	19.10	9.29	0.06	-0.40	-1.85
									μ	**0.09**	0.23	0.19
									σ	0.12	0.42	1.38

Table D.88: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=URIOrEmail, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	2.65	38.60	35.40	2.20	38.70	35.40	0.31	-0.01	-0.01
Arwiki	4	2	2.75	45.80	17.90	2.50	39.00	17.80	0.18	0.47	0.07
Arwiki	4	3	2.33	41.30	16.70	2.41	33.20	13.70	-0.05	0.57	2.12
Arwiki	4	4	2.47	33.60	12.80	2.43	28.80	11.80	0.03	0.33	0.69
Arwiki	16	1	2.27	38.60	35.40	2.12	38.70	35.40	0.10	-0.01	-0.01
Arwiki	16	2	2.23	29.80	17.50	2.19	26.90	17.40	0.03	0.21	0.06
Arwiki	16	3	2.00	23.80	14.70	1.80	25.30	16.80	0.14	-0.11	-1.45
Arwiki	16	4	1.98	20.20	12.70	2.09	17.20	10.80	-0.08	0.21	1.33
Arwiki	32	1	2.19	37.80	35.40	2.12	38.00	35.40	0.05	-0.01	-0.01
Arwiki	32	2	1.89	26.80	20.70	1.87	25.30	20.60	0.01	0.10	0.02
Arwiki	32	3	1.69	21.90	17.00	1.83	19.30	14.70	-0.10	0.18	1.56
Arwiki	32	4	1.85	16.90	13.10	1.95	14.60	10.90	-0.07	0.16	1.50
Enwiki	4	1	1.15	15.90	15.50	.84	15.90	15.50	0.31	-0.00	-0.00
Enwiki	4	2	.81	21.90	7.78	.60	18.40	7.77	0.21	0.36	0.01
Enwiki	4	3	.68	17.40	5.18	.48	15.40	5.18	0.20	0.20	0.00
Enwiki	4	4	.63	13.40	3.90	.45	12.00	3.88	0.19	0.14	0.02
Enwiki	16	1	.88	15.80	15.50	.78	15.90	15.50	0.10	-0.00	-0.00
Enwiki	16	2	.52	14.10	7.78	.47	11.30	7.77	0.05	0.28	0.01
Enwiki	16	3	.45	9.67	5.19	.38	9.67	5.18	0.07	0.00	0.01
Enwiki	16	4	.45	7.80	3.91	.32	6.60	3.88	0.14	0.12	0.02
Enwiki	32	1	.84	15.80	15.50	.79	15.80	15.50	0.05	0.00	0.00
Enwiki	32	2	.49	10.70	7.79	.47	9.27	7.76	0.03	0.15	0.03
Enwiki	32	3	.42	7.47	5.19	.31	7.76	5.17	0.11	-0.03	0.02
Enwiki	32	4	.43	5.88	3.92	.25	5.82	3.88	0.18	0.01	0.04
Ruwiki	4	1	3.21	45.60	42.00	2.64	45.60	42.10	0.32	-0.00	-0.00
Ruwiki	4	2	3.06	56.30	23.10	2.77	48.40	23.10	0.17	0.45	0.00
Ruwiki	4	3	2.67	50.30	20.70	2.74	40.40	17.10	-0.04	0.56	2.03
Ruwiki	4	4	2.85	40.30	15.30	2.83	34.60	13.80	0.01	0.32	0.82
Ruwiki	16	1	2.73	45.50	42.00	2.54	45.60	42.10	0.10	-0.01	-0.02
Ruwiki	16	2	2.30	39.60	24.70	2.23	36.20	24.70	0.04	0.19	0.01
Ruwiki	16	3	2.15	30.60	19.40	2.39	26.10	15.80	-0.14	0.25	2.05
Ruwiki	16	4	2.20	24.90	15.80	2.33	21.80	13.60	-0.08	0.17	1.24
Ruwiki	32	1	2.64	44.80	42.00	2.55	44.90	42.00	0.05	-0.01	-0.00
Ruwiki	32	2	2.40	29.20	21.70	2.37	27.80	21.60	0.02	0.08	0.02
Ruwiki	32	3	2.22	22.20	16.40	2.24	21.30	15.80	-0.01	0.05	0.34
Ruwiki	32	4	1.98	21.40	16.80	2.17	18.30	13.60	-0.10	0.17	1.81
Zhwiki	4	1	1.32	19.00	17.20	1.11	19.00	17.20	0.31	-0.00	0.01
Zhwiki	4	2	1.14	25.10	11.80	1.02	22.10	11.80	0.17	0.44	-0.01
Zhwiki	4	3	1.28	18.80	7.14	1.25	16.00	6.64	0.04	0.41	0.74
Zhwiki	4	4	1.23	16.70	6.58	1.27	13.90	5.61	-0.07	0.41	1.42
Zhwiki	16	1	1.14	18.90	17.20	1.07	19.00	17.20	0.10	-0.01	0.00
Zhwiki	16	2	.89	17.50	11.50	.87	16.30	11.50	0.02	0.18	0.01
Zhwiki	16	3	1.04	11.50	6.99	.78	14.30	10.00	0.39	-0.42	-4.47
Zhwiki	16	4	.99	10.30	6.58	.97	9.97	6.64	0.03	0.05	-0.10
Zhwiki	32	1	1.11	18.60	17.20	1.07	18.70	17.30	0.05	-0.01	-0.04
Zhwiki	32	2	.97	13.00	9.79	.97	12.50	9.78	-0.00	0.08	0.01
Zhwiki	32	3	1.01	9.13	6.74	.77	11.90	9.54	0.36	-0.41	-4.12
Zhwiki	32	4	.91	8.75	6.82	.86	9.15	7.08	0.07	-0.06	-0.39
								μ	**0.08**	0.13	0.15
								σ	0.13	0.20	1.17

Table D.89: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=URIOrEmail, Colours=never

Arguments: Xquote -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	5.65	16.00	7.68	4.87	15.80	7.67	0.55	0.01	0.01
Arwiki	4	2	3.78	17.30	4.22	3.33	16.70	3.59	0.32	0.04	0.44
Arwiki	4	3	3.23	15.70	3.52	2.61	18.70	5.19	0.43	-0.20	-1.17
Arwiki	4	4	3.41	11.30	2.67	2.79	12.90	2.99	0.43	-0.11	-0.22
Arwiki	16	1	4.95	16.50	7.96	4.70	16.00	7.73	0.18	0.03	0.16
Arwiki	16	2	3.27	13.20	5.24	2.73	14.80	5.82	0.38	-0.11	-0.40
Arwiki	16	3	2.82	10.30	3.95	2.36	14.00	5.36	0.32	-0.26	-0.98
Arwiki	16	4	2.91	8.73	3.39	2.62	9.19	3.31	0.20	-0.03	0.06
Arwiki	32	1	4.87	17.20	8.03	4.72	16.60	8.00	0.10	0.04	0.02
Arwiki	32	2	3.47	10.90	4.32	2.94	11.00	4.44	0.37	-0.00	-0.08
Arwiki	32	3	2.68	10.20	4.18	2.36	12.90	5.22	0.23	-0.19	-0.73
Arwiki	32	4	2.86	8.15	3.39	2.71	7.33	2.77	0.10	0.06	0.44
Enwiki	4	1	3.24	3.48	1.57	2.76	3.41	1.65	0.49	0.01	-0.08
Enwiki	4	2	1.86	6.57	.61	1.57	6.66	.65	0.29	-0.01	-0.04
Enwiki	4	3	1.71	5.55	.54	1.47	5.63	.70	0.24	-0.01	-0.16
Enwiki	4	4	1.77	3.84	.23	1.45	4.26	.33	0.33	-0.04	-0.09
Enwiki	16	1	2.77	3.27	1.53	2.65	3.23	1.53	0.12	0.00	-0.01
Enwiki	16	2	1.58	3.00	.62	1.44	3.01	.65	0.15	-0.00	-0.02
Enwiki	16	3	1.50	2.58	.68	1.37	3.17	.73	0.13	-0.06	-0.04
Enwiki	16	4	1.57	1.62	.26	1.37	2.24	.30	0.20	-0.06	-0.04
Enwiki	32	1	2.73	3.46	1.64	2.67	3.32	1.59	0.06	0.01	0.05
Enwiki	32	2	1.49	2.47	.63	1.43	2.58	.65	0.07	-0.01	-0.02
Enwiki	32	3	1.50	1.94	.63	1.35	2.90	.78	0.15	-0.10	-0.15
Enwiki	32	4	1.55	1.29	.29	1.35	1.90	.33	0.20	-0.06	-0.04
Ruwiki	4	1	6.97	18.10	8.64	6.03	17.50	8.33	0.53	0.03	0.17
Ruwiki	4	2	4.34	23.00	5.18	3.64	23.50	6.09	0.40	-0.02	-0.51
Ruwiki	4	3	3.91	17.50	4.09	3.39	17.70	3.99	0.30	-0.01	0.06
Ruwiki	4	4	3.98	14.10	3.36	3.18	17.00	4.30	0.45	-0.16	-0.53
Ruwiki	16	1	6.10	18.50	8.89	5.79	17.80	8.40	0.18	0.04	0.28
Ruwiki	16	2	3.81	14.70	5.37	3.46	14.50	5.36	0.20	0.01	0.00
Ruwiki	16	3	3.49	11.10	4.25	3.06	12.80	4.41	0.25	-0.09	-0.09
Ruwiki	16	4	3.48	9.77	3.68	3.21	9.05	2.96	0.15	0.04	0.41
Ruwiki	32	1	6.00	19.40	9.13	5.82	18.50	8.85	0.10	0.05	0.16
Ruwiki	32	2	3.92	11.10	4.16	3.68	10.80	4.08	0.14	0.02	0.04
Ruwiki	32	3	3.49	9.55	3.75	3.14	10.20	3.62	0.20	-0.03	0.08
Ruwiki	32	4	3.57	8.36	3.43	3.17	8.49	3.12	0.23	-0.01	0.17
Zhwiki	4	1	2.91	7.78	3.73	2.56	7.48	3.57	0.51	0.04	0.23
Zhwiki	4	2	1.83	9.12	2.09	1.71	7.59	1.62	0.19	0.23	0.69
Zhwiki	4	3	1.70	6.59	1.68	1.40	7.67	1.85	0.43	-0.16	-0.24
Zhwiki	4	4	1.68	5.11	1.30	1.43	6.09	1.36	0.37	-0.14	-0.08
Zhwiki	16	1	2.57	7.89	3.83	2.48	7.57	3.61	0.14	0.05	0.32
Zhwiki	16	2	1.63	6.25	2.37	1.47	6.40	2.44	0.23	-0.02	-0.10
Zhwiki	16	3	1.51	4.20	1.63	1.33	4.53	1.51	0.27	-0.05	0.18
Zhwiki	16	4	1.52	3.54	1.33	1.19	5.58	2.14	0.49	-0.30	-1.18
Zhwiki	32	1	2.53	8.22	3.88	2.48	7.86	3.74	0.06	0.05	0.20
Zhwiki	32	2	1.49	6.93	2.86	1.38	7.08	3.01	0.15	-0.02	-0.22
Zhwiki	32	3	1.50	3.74	1.49	1.33	3.97	1.42	0.25	-0.03	0.11
Zhwiki	32	4	1.37	4.38	1.86	1.29	3.76	1.43	0.12	0.09	0.62
								μ	**0.26**	-0.03	-0.05
								σ	0.14	0.09	0.38

Table D.90: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=Xquote, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	3.95	77.40	46.80	3.26	73.80	47.00	0.48	0.25	-0.14
Arwiki	4	2	2.68	98.70	26.40	2.51	79.10	23.10	0.12	1.37	2.30
Arwiki	4	3	2.32	83.90	20.80	2.06	78.30	23.60	0.18	0.39	-1.95
Arwiki	4	4	2.38	64.40	14.60	2.18	58.10	15.10	0.14	0.44	-0.32
Arwiki	16	1	3.39	80.60	47.20	3.16	76.60	47.20	0.16	0.28	-0.01
Arwiki	16	2	2.24	65.30	26.50	2.02	64.30	30.50	0.15	0.07	-2.83
Arwiki	16	3	1.93	48.30	20.00	1.80	55.80	24.10	0.10	-0.53	-2.83
Arwiki	16	4	1.99	39.60	16.00	1.97	39.10	16.20	0.02	0.03	-0.07
Arwiki	32	1	3.30	81.90	47.20	3.17	76.00	47.40	0.09	0.41	-0.14
Arwiki	32	2	2.15	54.70	27.10	2.18	50.90	25.10	-0.02	0.27	1.44
Arwiki	32	3	1.85	43.20	21.30	1.77	50.70	23.90	0.06	-0.53	-1.79
Arwiki	32	4	1.92	33.30	16.50	2.01	32.10	13.90	-0.06	0.08	1.87
Enwiki	4	1	2.11	34.60	22.00	1.67	31.70	22.10	0.45	0.29	-0.05
Enwiki	4	2	1.22	44.40	11.10	1.00	40.40	11.00	0.22	0.40	0.06
Enwiki	4	3	1.10	36.60	7.42	.92	35.00	7.28	0.18	0.16	0.13
Enwiki	4	4	1.11	28.30	5.58	.90	26.90	5.51	0.21	0.14	0.07
Enwiki	16	1	1.73	34.10	22.20	1.60	32.50	22.20	0.13	0.16	0.01
Enwiki	16	2	.93	31.10	11.20	.87	26.10	11.10	0.07	0.51	0.07
Enwiki	16	3	.84	21.50	7.39	.80	22.50	7.27	0.03	-0.10	0.12
Enwiki	16	4	.84	16.90	5.54	.80	16.60	5.53	0.03	0.04	0.01
Enwiki	32	1	1.68	34.70	22.30	1.61	32.20	22.20	0.07	0.25	0.06
Enwiki	32	2	.89	22.70	11.20	.86	22.70	11.10	0.04	0.00	0.04
Enwiki	32	3	.79	16.80	7.38	.79	20.10	7.26	0.01	-0.33	0.12
Enwiki	32	4	.81	13.00	5.54	.79	14.90	5.54	0.02	-0.20	-0.00
Ruwiki	4	1	4.83	94.80	54.90	3.99	86.30	55.20	0.48	0.48	-0.14
Ruwiki	4	2	3.18	115.00	31.00	2.71	102.00	34.50	0.26	0.72	-1.98
Ruwiki	4	3	2.78	95.20	22.50	2.65	79.30	19.70	0.07	0.90	1.58
Ruwiki	4	4	2.71	77.60	18.40	2.38	73.30	21.20	0.19	0.24	-1.61
Ruwiki	16	1	4.14	95.90	55.20	3.86	89.20	55.40	0.16	0.38	-0.12
Ruwiki	16	2	2.63	77.60	31.10	2.51	69.50	31.60	0.07	0.46	-0.24
Ruwiki	16	3	2.26	56.10	22.50	2.23	57.60	22.70	0.02	-0.08	-0.10
Ruwiki	16	4	2.22	48.80	19.70	2.33	42.20	16.20	-0.06	0.37	1.98
Ruwiki	32	1	4.03	98.20	55.30	3.86	89.60	55.50	0.09	0.49	-0.09
Ruwiki	32	2	2.56	64.00	30.60	2.64	56.40	26.70	-0.04	0.43	2.18
Ruwiki	32	3	2.24	45.20	20.60	2.29	46.90	19.40	-0.03	-0.10	0.63
Ruwiki	32	4	2.20	38.10	18.30	2.29	38.30	16.20	-0.05	-0.01	1.19
Zhwiki	4	1	2.02	37.70	22.20	1.70	35.30	22.20	0.47	0.35	-0.03
Zhwiki	4	2	1.36	44.60	12.20	1.27	36.30	10.80	0.14	1.21	2.04
Zhwiki	4	3	1.18	37.40	9.29	1.08	34.20	9.26	0.15	0.48	0.04
Zhwiki	4	4	1.17	29.70	7.33	1.08	27.50	7.00	0.13	0.33	0.49
Zhwiki	16	1	1.75	38.40	22.30	1.64	36.90	22.30	0.15	0.22	0.03
Zhwiki	16	2	1.15	30.60	12.40	1.07	29.30	13.40	0.12	0.20	-1.48
Zhwiki	16	3	.99	22.60	9.37	1.00	21.20	7.95	-0.02	0.21	2.09
Zhwiki	16	4	1.01	18.30	7.44	.89	22.00	9.60	0.17	-0.55	-3.16
Zhwiki	32	1	1.70	38.70	22.40	1.65	36.50	22.30	0.08	0.32	0.14
Zhwiki	32	2	1.16	24.60	11.60	1.00	29.30	14.90	0.23	-0.69	-4.87
Zhwiki	32	3	.97	18.60	8.69	.99	18.20	7.69	-0.03	0.06	1.46
Zhwiki	32	4	.93	17.30	8.23	.96	16.10	7.00	-0.05	0.17	1.82
								μ	**0.12**	0.22	-0.04
								σ	0.14	0.39	1.47

Table D.91: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=Xquote, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	3.10	38.20	35.00	2.56	38.20	35.00	0.38	-0.01	-0.01
Arwiki	4	2	3.00	48.20	17.30	2.75	41.30	17.30	0.18	0.48	0.00
Arwiki	4	3	2.79	40.80	12.80	2.17	40.00	18.00	0.43	0.05	-3.61
Arwiki	4	4	2.60	35.30	11.90	2.51	30.00	11.50	0.06	0.37	0.24
Arwiki	16	1	2.64	38.00	35.00	2.47	38.10	35.00	0.12	-0.01	-0.01
Arwiki	16	2	1.93	37.50	23.10	1.87	33.30	23.10	0.04	0.29	0.00
Arwiki	16	3	2.21	23.60	12.70	1.73	28.00	18.50	0.33	-0.31	-4.06
Arwiki	16	4	1.94	21.50	12.80	2.00	19.50	12.30	-0.04	0.14	0.32
Arwiki	32	1	2.56	37.40	35.00	2.48	37.50	35.00	0.06	-0.01	-0.01
Arwiki	32	2	2.19	25.30	18.60	2.16	23.90	18.60	0.02	0.10	0.01
Arwiki	32	3	2.02	18.70	13.40	1.63	23.60	18.30	0.27	-0.34	-3.47
Arwiki	32	4	1.91	16.30	12.10	2.03	14.80	10.50	-0.09	0.11	1.14
Enwiki	4	1	1.49	16.10	15.60	1.12	16.10	15.60	0.37	-0.00	-0.00
Enwiki	4	2	1.02	24.10	7.85	.80	20.00	7.85	0.22	0.41	0.00
Enwiki	4	3	.86	19.50	5.24	.67	17.40	5.22	0.20	0.21	0.03
Enwiki	4	4	.82	15.20	3.96	.64	13.10	3.96	0.19	0.21	0.00
Enwiki	16	1	1.17	16.00	15.60	1.05	16.00	15.60	0.11	-0.00	-0.00
Enwiki	16	2	.69	15.90	7.85	.64	12.30	7.84	0.05	0.36	0.01
Enwiki	16	3	.52	10.80	5.24	.52	11.70	5.22	0.00	-0.09	0.02
Enwiki	16	4	.47	9.01	3.96	.43	8.20	3.95	0.05	0.08	0.02
Enwiki	32	1	1.12	16.00	15.60	1.06	16.00	15.60	0.06	0.00	-0.00
Enwiki	32	2	.64	11.50	7.85	.61	9.63	7.83	0.03	0.18	0.02
Enwiki	32	3	.43	7.98	5.24	.43	9.18	5.23	0.01	-0.12	0.02
Enwiki	32	4	.44	6.35	3.97	.40	6.82	3.95	0.04	-0.05	0.02
Ruwiki	4	1	3.74	44.10	40.80	3.06	44.20	40.80	0.39	-0.01	0.01
Ruwiki	4	2	2.99	61.90	26.00	2.64	53.00	26.00	0.20	0.50	-0.00
Ruwiki	4	3	2.90	50.80	18.20	2.89	40.00	15.00	0.01	0.61	1.86
Ruwiki	4	4	2.88	41.20	14.60	2.54	37.40	16.20	0.20	0.22	-0.88
Ruwiki	16	1	3.16	44.00	40.80	2.95	44.00	40.80	0.12	-0.00	0.01
Ruwiki	16	2	2.44	40.60	23.60	2.39	35.70	23.60	0.03	0.28	0.01
Ruwiki	16	3	2.15	32.10	18.90	2.25	28.60	17.10	-0.06	0.20	1.00
Ruwiki	16	4	2.16	25.40	14.80	2.34	21.00	12.20	-0.10	0.25	1.44
Ruwiki	32	1	3.07	43.40	40.80	2.95	43.50	40.80	0.07	-0.01	-0.01
Ruwiki	32	2	2.65	27.50	19.60	2.61	26.00	19.60	0.02	0.09	0.02
Ruwiki	32	3	2.10	23.50	17.20	1.82	27.10	20.60	0.16	-0.21	-1.95
Ruwiki	32	4	2.15	18.00	13.10	2.20	17.50	12.20	-0.03	0.03	0.51
Zhwiki	4	1	1.57	18.20	16.60	1.31	18.20	16.60	0.38	-0.01	-0.01
Zhwiki	4	2	1.49	22.10	8.05	1.37	18.80	8.02	0.18	0.48	0.04
Zhwiki	4	3	1.23	20.60	7.73	1.18	17.20	7.10	0.07	0.50	0.92
Zhwiki	4	4	1.28	16.50	5.63	1.22	13.90	5.36	0.09	0.38	0.39
Zhwiki	16	1	1.34	18.20	16.60	1.27	18.20	16.60	0.12	-0.00	-0.01
Zhwiki	16	2	1.04	17.00	10.10	1.01	15.10	10.10	0.03	0.29	0.01
Zhwiki	16	3	.93	13.30	8.05	1.09	10.40	5.94	-0.23	0.42	3.10
Zhwiki	16	4	1.02	9.54	5.35	.85	11.00	7.38	0.25	-0.21	-2.97
Zhwiki	32	1	1.30	17.90	16.60	1.27	18.00	16.60	0.06	-0.01	-0.01
Zhwiki	32	2	.89	14.60	11.20	.88	14.20	11.30	0.01	0.07	-0.18
Zhwiki	32	3	1.00	8.92	6.26	1.05	8.10	5.72	-0.08	0.12	0.79
Zhwiki	32	4	.89	8.18	6.08	.97	7.45	5.29	-0.12	0.11	1.16
								μ	**0.10**	0.13	-0.08
								σ	0.14	0.22	1.25

Table D.92: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=Xquote, Colours=never

Arguments: \p{Greek} -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	3.18	4.88	2.23	2.42	4.70	2.29	0.53	0.01	-0.05
Arwiki	4	2	1.80	9.87	.79	1.39	9.43	.83	0.29	0.03	-0.02
Arwiki	4	3	1.43	8.00	.74	1.04	8.14	.83	0.27	-0.01	-0.07
Arwiki	4	4	1.23	5.87	.34	.90	7.64	.39	0.23	-0.12	-0.04
Arwiki	16	1	2.50	4.53	2.12	2.27	4.47	2.08	0.16	0.00	0.03
Arwiki	16	2	1.38	4.53	.84	1.27	5.14	1.04	0.08	-0.04	-0.14
Arwiki	16	3	1.18	3.15	.94	.88	4.50	.89	0.21	-0.09	0.04
Arwiki	16	4	1.05	2.26	.37	.78	4.59	.43	0.18	-0.16	-0.04
Arwiki	32	1	2.41	4.64	2.16	2.27	4.45	2.09	0.10	0.01	0.05
Arwiki	32	2	1.40	3.17	.88	1.25	4.29	1.15	0.10	-0.08	-0.19
Arwiki	32	3	1.03	2.60	.88	.80	3.05	.92	0.16	-0.03	-0.03
Arwiki	32	4	1.04	1.73	.38	.66	2.90	.60	0.27	-0.08	-0.16
Enwiki	4	1	1.93	3.35	1.51	1.46	3.15	1.52	0.47	0.02	-0.02
Enwiki	4	2	1.10	6.55	.54	.84	6.16	.57	0.27	0.04	-0.02
Enwiki	4	3	.88	5.86	.52	.61	5.15	.57	0.27	0.07	-0.05
Enwiki	4	4	.81	4.26	.23	.49	4.31	.29	0.32	-0.00	-0.07
Enwiki	16	1	1.47	3.02	1.42	1.36	2.96	1.38	0.11	0.01	0.04
Enwiki	16	2	.85	2.49	.56	.74	3.05	.70	0.12	-0.06	-0.14
Enwiki	16	3	.75	2.27	.68	.50	2.68	.62	0.26	-0.04	0.06
Enwiki	16	4	.70	1.57	.24	.46	2.53	.42	0.25	-0.10	-0.18
Enwiki	32	1	1.42	3.18	1.53	1.37	3.01	1.42	0.05	0.02	0.11
Enwiki	32	2	.80	2.21	.57	.73	2.49	.82	0.07	-0.03	-0.25
Enwiki	32	3	.72	1.78	.59	.48	1.95	.66	0.24	-0.02	-0.07
Enwiki	32	4	.71	1.28	.25	.43	1.75	.42	0.28	-0.05	-0.17
Ruwiki	4	1	4.16	6.75	3.16	3.25	6.55	3.24	0.51	0.01	-0.05
Ruwiki	4	2	2.29	14.20	1.24	1.89	12.60	1.28	0.23	0.09	-0.02
Ruwiki	4	3	1.82	10.50	1.06	1.41	10.70	1.20	0.23	-0.01	-0.08
Ruwiki	4	4	1.59	7.48	.55	1.34	11.40	.61	0.14	-0.22	-0.03
Ruwiki	16	1	3.31	6.28	2.99	3.04	6.29	2.99	0.15	-0.00	0.00
Ruwiki	16	2	1.91	5.31	1.22	1.74	7.20	1.51	0.10	-0.11	-0.17
Ruwiki	16	3	1.41	4.58	1.38	1.20	6.22	1.27	0.12	-0.09	0.07
Ruwiki	16	4	1.39	3.06	.55	1.06	6.10	.71	0.19	-0.17	-0.09
Ruwiki	32	1	3.20	6.59	3.15	3.05	6.29	2.97	0.09	0.02	0.10
Ruwiki	32	2	1.94	4.25	1.30	1.69	6.21	1.78	0.14	-0.11	-0.27
Ruwiki	32	3	1.36	3.56	1.25	1.09	4.34	1.37	0.15	-0.04	-0.06
Ruwiki	32	4	1.34	2.49	.68	.92	4.20	.88	0.24	-0.10	-0.12
Zhwiki	4	1	1.83	2.53	1.16	1.49	2.41	1.21	0.50	0.02	-0.07
Zhwiki	4	2	1.03	4.62	.44	.85	4.59	.45	0.27	0.00	-0.02
Zhwiki	4	3	.79	3.88	.38	.63	4.13	.47	0.23	-0.04	-0.13
Zhwiki	4	4	.68	2.94	.20	.63	5.09	.24	0.07	-0.32	-0.07
Zhwiki	16	1	1.50	2.31	1.10	1.40	2.27	1.09	0.14	0.01	0.00
Zhwiki	16	2	.86	2.00	.41	.79	2.74	.51	0.09	-0.11	-0.14
Zhwiki	16	3	.63	1.64	.49	.57	2.66	.47	0.09	-0.15	0.02
Zhwiki	16	4	.56	1.17	.19	.49	2.72	.25	0.10	-0.23	-0.08
Zhwiki	32	1	1.45	2.41	1.16	1.40	2.31	1.12	0.07	0.02	0.06
Zhwiki	32	2	.82	1.57	.47	.78	2.31	.55	0.05	-0.11	-0.13
Zhwiki	32	3	.56	1.39	.50	.50	1.66	.51	0.08	-0.04	-0.01
Zhwiki	32	4	.53	.91	.23	.43	1.70	.29	0.14	-0.12	-0.09
								μ	**0.20**	-0.05	-0.06
								σ	0.12	0.08	0.09

Table D.93: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=\p{Greek}, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	2.08	46.00	31.70	1.42	45.20	32.00	0.47	0.05	-0.19
Arwiki	4	2	1.24	64.20	16.00	.89	53.80	15.80	0.25	0.72	0.10
Arwiki	4	3	.94	54.10	10.60	.66	45.40	10.50	0.20	0.61	0.09
Arwiki	4	4	.83	43.30	7.91	.60	41.70	7.85	0.16	0.12	0.04
Arwiki	16	1	1.57	46.80	32.00	1.34	45.50	32.10	0.16	0.09	-0.08
Arwiki	16	2	.88	41.00	16.10	.76	35.40	16.00	0.09	0.39	0.11
Arwiki	16	3	.70	28.20	10.60	.54	28.40	10.50	0.11	-0.02	0.03
Arwiki	16	4	.70	23.50	7.93	.48	24.80	7.95	0.15	-0.09	-0.02
Arwiki	32	1	1.48	46.60	32.10	1.35	44.80	32.10	0.09	0.13	0.00
Arwiki	32	2	.81	31.40	16.10	.73	28.90	16.00	0.05	0.18	0.08
Arwiki	32	3	.67	22.10	10.60	.47	21.40	10.50	0.14	0.05	0.03
Arwiki	32	4	.68	17.40	7.98	.40	18.20	7.95	0.20	-0.06	0.02
Enwiki	4	1	1.38	32.00	21.90	.95	31.50	22.10	0.43	0.05	-0.19
Enwiki	4	2	.82	46.20	11.00	.60	37.10	10.90	0.22	0.92	0.08
Enwiki	4	3	.67	35.20	7.35	.44	31.10	7.23	0.23	0.41	0.12
Enwiki	4	4	.60	27.60	5.50	.37	25.90	5.43	0.23	0.17	0.07
Enwiki	16	1	1.01	32.70	22.20	.89	31.60	22.20	0.12	0.11	0.01
Enwiki	16	2	.57	28.40	11.20	.49	23.60	11.00	0.08	0.49	0.16
Enwiki	16	3	.50	19.00	7.40	.34	18.20	7.30	0.16	0.07	0.11
Enwiki	16	4	.48	15.50	5.52	.29	14.50	5.49	0.19	0.10	0.03
Enwiki	32	1	.96	32.50	22.20	.90	31.30	22.10	0.06	0.12	0.06
Enwiki	32	2	.53	21.60	11.10	.48	19.40	11.10	0.04	0.22	0.07
Enwiki	32	3	.47	15.70	7.37	.31	14.30	7.30	0.15	0.14	0.07
Enwiki	32	4	.48	12.20	5.54	.26	11.70	5.49	0.22	0.06	0.05
Ruwiki	4	1	2.80	60.20	39.50	1.98	57.20	39.60	0.46	0.17	-0.03
Ruwiki	4	2	1.67	78.70	20.00	1.24	67.70	19.90	0.24	0.62	0.05
Ruwiki	4	3	1.28	65.30	13.40	.94	57.70	13.10	0.19	0.43	0.18
Ruwiki	4	4	1.09	52.70	10.10	.88	54.70	9.86	0.12	-0.12	0.11
Ruwiki	16	1	2.14	59.70	39.80	1.87	57.80	39.80	0.15	0.11	0.00
Ruwiki	16	2	1.23	50.60	20.10	1.07	45.10	19.90	0.09	0.31	0.10
Ruwiki	16	3	.98	34.50	13.50	.78	37.10	13.20	0.11	-0.14	0.15
Ruwiki	16	4	.90	29.30	10.10	.68	32.10	9.98	0.12	-0.15	0.05
Ruwiki	32	1	2.02	60.30	39.80	1.87	57.20	39.90	0.09	0.17	-0.03
Ruwiki	32	2	1.22	40.50	20.20	1.05	37.70	20.00	0.10	0.16	0.09
Ruwiki	32	3	.91	29.70	13.40	.68	27.90	13.20	0.13	0.10	0.12
Ruwiki	32	4	.85	22.00	10.00	.58	24.20	10.00	0.15	-0.13	0.01
Zhwiki	4	1	1.20	22.40	15.20	.87	22.00	15.20	0.48	0.06	-0.01
Zhwiki	4	2	.69	31.80	7.68	.54	26.20	7.59	0.23	0.82	0.13
Zhwiki	4	3	.53	25.10	5.17	.40	22.10	5.03	0.18	0.44	0.20
Zhwiki	4	4	.43	20.90	3.82	.38	20.10	3.79	0.08	0.12	0.05
Zhwiki	16	1	.94	22.70	15.30	.83	22.30	15.30	0.17	0.05	-0.00
Zhwiki	16	2	.51	20.70	7.70	.47	17.30	7.67	0.06	0.49	0.05
Zhwiki	16	3	.38	13.80	5.12	.34	14.70	5.08	0.05	-0.12	0.06
Zhwiki	16	4	.35	11.20	3.86	.30	12.50	3.82	0.08	-0.19	0.06
Zhwiki	32	1	.90	22.60	15.30	.83	21.90	15.30	0.10	0.09	0.03
Zhwiki	32	2	.49	15.30	7.71	.46	13.90	7.68	0.05	0.20	0.05
Zhwiki	32	3	.35	10.80	5.14	.30	10.70	5.06	0.07	0.02	0.11
Zhwiki	32	4	.34	8.75	3.88	.26	9.12	3.83	0.11	-0.05	0.07
								μ	**0.16**	0.18	0.05
								σ	0.11	0.25	0.08

Table D.94: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=\p{Greek}, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	1.65	23.10	21.70	1.12	23.10	22.40	0.37	-0.00	-0.51
Arwiki	4	2	1.19	34.60	11.30	.86	28.00	11.30	0.23	0.46	0.00
Arwiki	4	3	1.00	27.80	7.52	.67	21.70	7.52	0.23	0.42	0.00
Arwiki	4	4	.93	22.50	5.66	.61	17.60	5.64	0.22	0.34	0.01
Arwiki	16	1	1.21	23.10	22.40	1.04	23.10	22.40	0.12	-0.00	0.00
Arwiki	16	2	.73	22.10	11.30	.65	17.20	11.20	0.06	0.34	0.01
Arwiki	16	3	.65	14.90	7.52	.52	13.70	7.51	0.09	0.08	0.00
Arwiki	16	4	.66	12.00	5.67	.46	10.40	5.63	0.14	0.12	0.03
Arwiki	32	1	1.14	23.00	22.40	1.05	23.00	22.40	0.06	0.00	0.00
Arwiki	32	2	.65	15.80	11.30	.62	13.70	11.20	0.02	0.15	0.02
Arwiki	32	3	.61	10.90	7.52	.42	10.40	7.51	0.14	0.04	0.01
Arwiki	32	4	.62	8.45	5.67	.36	8.16	5.63	0.18	0.02	0.02
Enwiki	4	1	1.16	16.00	15.10	.79	16.00	15.50	0.37	-0.00	-0.41
Enwiki	4	2	.82	23.70	7.78	.60	19.30	7.78	0.23	0.44	0.01
Enwiki	4	3	.69	18.80	5.20	.47	15.00	5.19	0.23	0.38	0.01
Enwiki	4	4	.64	15.10	3.91	.41	12.00	3.89	0.23	0.31	0.01
Enwiki	16	1	.84	15.90	15.50	.73	15.90	15.50	0.12	-0.00	0.00
Enwiki	16	2	.50	15.10	7.78	.44	11.80	7.77	0.06	0.34	0.01
Enwiki	16	3	.45	9.94	5.19	.34	9.24	5.19	0.10	0.07	0.01
Enwiki	16	4	.45	8.21	3.92	.30	6.70	3.89	0.15	0.15	0.03
Enwiki	32	1	.79	15.90	15.50	.73	15.90	15.50	0.06	-0.00	0.00
Enwiki	32	2	.47	10.90	7.80	.42	9.40	7.77	0.05	0.15	0.03
Enwiki	32	3	.42	7.52	5.20	.28	7.13	5.18	0.14	0.04	0.01
Enwiki	32	4	.43	5.90	3.92	.24	5.47	3.89	0.20	0.04	0.04
Ruwiki	4	1	2.25	28.90	27.10	1.59	29.00	28.00	0.37	-0.00	-0.48
Ruwiki	4	2	1.58	43.60	14.20	1.20	35.20	14.20	0.21	0.48	0.01
Ruwiki	4	3	1.31	34.70	9.49	.96	27.30	9.42	0.20	0.42	0.04
Ruwiki	4	4	1.19	28.00	7.19	.95	23.00	7.11	0.14	0.28	0.04
Ruwiki	16	1	1.69	28.90	27.90	1.48	28.90	27.90	0.12	0.00	0.00
Ruwiki	16	2	1.03	28.10	14.10	.94	22.10	14.10	0.05	0.34	0.02
Ruwiki	16	3	.84	18.90	9.49	.78	17.80	9.47	0.03	0.07	0.01
Ruwiki	16	4	.86	15.30	7.19	.67	13.80	7.13	0.11	0.09	0.03
Ruwiki	32	1	1.59	28.80	28.00	1.48	28.80	27.90	0.06	0.00	0.00
Ruwiki	32	2	.92	20.10	14.20	.92	17.70	14.10	0.00	0.14	0.02
Ruwiki	32	3	.78	14.00	9.49	.61	13.60	9.47	0.10	0.02	0.01
Ruwiki	32	4	.81	10.80	7.18	.55	10.80	7.13	0.15	-0.00	0.03
Zhwiki	4	1	.92	11.10	10.50	.67	11.10	10.70	0.37	-0.00	-0.38
Zhwiki	4	2	.64	16.70	5.40	.49	13.40	5.40	0.21	0.48	0.01
Zhwiki	4	3	.51	13.00	3.63	.38	10.40	3.61	0.18	0.38	0.03
Zhwiki	4	4	.45	10.60	2.75	.36	8.56	2.73	0.14	0.31	0.03
Zhwiki	16	1	.70	11.10	10.70	.62	11.10	10.70	0.12	-0.00	0.02
Zhwiki	16	2	.40	10.80	5.42	.38	8.45	5.40	0.03	0.34	0.02
Zhwiki	16	3	.32	7.21	3.63	.31	6.85	3.64	0.02	0.05	-0.01
Zhwiki	16	4	.32	5.93	2.76	.26	5.30	2.74	0.09	0.09	0.03
Zhwiki	32	1	.66	11.10	10.70	.62	11.10	10.70	0.06	-0.00	0.02
Zhwiki	32	2	.37	7.64	5.42	.39	6.79	5.40	-0.02	0.12	0.02
Zhwiki	32	3	.30	5.37	3.63	.25	5.27	3.63	0.07	0.01	0.00
Zhwiki	32	4	.30	4.28	2.76	.22	4.10	2.73	0.11	0.03	0.05
								μ	**0.14**	0.16	-0.02
								σ	0.10	0.17	0.13

Table D.95: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=\p{Greek}, Colours=never

Arguments: \X -colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	68.30	104.00	24.70	67.20	91.20	24.70	0.77	0.88	-0.00
Arwiki	4	2	58.60	70.90	16.10	56.30	68.10	16.30	1.59	0.19	-0.14
Arwiki	4	3	58.80	54.40	14.00	55.40	66.10	20.30	2.37	-0.81	-4.45
Arwiki	4	4	58.10	46.70	12.10	56.30	45.70	13.70	1.29	0.07	-1.17
Arwiki	16	1	65.10	77.50	27.40	64.90	76.60	26.80	0.14	0.06	0.39
Arwiki	16	2	56.30	44.30	14.70	55.70	42.90	14.40	0.40	0.10	0.17
Arwiki	16	3	56.20	43.10	16.10	54.80	51.00	20.00	1.01	-0.55	-2.74
Arwiki	16	4	57.10	33.60	12.70	56.10	30.10	11.40	0.71	0.25	0.93
Arwiki	32	1	64.60	72.30	25.90	64.50	73.20	26.00	0.08	-0.06	-0.05
Arwiki	32	2	55.40	43.20	15.80	55.10	43.90	16.00	0.17	-0.05	-0.15
Arwiki	32	3	57.10	35.10	13.00	55.90	30.20	10.80	0.81	0.34	1.56
Arwiki	32	4	57.00	32.10	12.80	56.00	29.00	11.40	0.73	0.22	0.96
Enwiki	4	1	46.40	53.80	14.60	45.60	55.10	14.60	0.84	-0.12	-0.04
Enwiki	4	2	38.40	50.80	12.80	37.80	48.20	13.30	0.52	0.26	-0.55
Enwiki	4	3	39.70	32.00	7.88	39.00	28.90	7.10	0.71	0.30	0.79
Enwiki	4	4	39.80	27.60	7.13	37.90	36.90	12.80	1.89	-0.94	-5.70
Enwiki	16	1	44.30	46.20	16.50	44.20	47.30	16.30	0.02	-0.11	0.22
Enwiki	16	2	38.10	37.10	13.70	37.20	38.30	14.50	0.88	-0.13	-0.83
Enwiki	16	3	38.90	23.80	8.55	38.30	22.10	7.76	0.58	0.17	0.80
Enwiki	16	4	39.30	20.70	7.73	37.40	31.60	13.20	1.89	-1.10	-5.56
Enwiki	32	1	43.90	43.80	15.40	44.00	45.10	15.30	-0.15	-0.13	0.09
Enwiki	32	2	37.70	34.10	13.20	37.00	35.90	13.90	0.71	-0.17	-0.72
Enwiki	32	3	38.40	22.20	8.51	38.20	20.20	7.37	0.23	0.20	1.15
Enwiki	32	4	38.90	19.00	7.53	37.20	30.30	13.10	1.74	-1.15	-5.57
Ruwiki	4	1	82.50	119.00	27.40	81.10	94.10	27.60	0.78	1.43	-0.11
Ruwiki	4	2	70.00	94.70	21.90	67.60	87.00	23.00	1.33	0.44	-0.63
Ruwiki	4	3	70.40	67.90	16.70	68.90	66.10	18.40	0.87	0.10	-0.97
Ruwiki	4	4	70.70	57.00	14.90	68.60	58.10	18.00	1.17	-0.06	-1.79
Ruwiki	16	1	78.60	90.70	31.00	78.10	86.40	30.70	0.31	0.24	0.14
Ruwiki	16	2	68.70	56.40	18.80	67.70	55.00	19.20	0.58	0.08	-0.22
Ruwiki	16	3	68.50	50.10	18.40	67.00	58.70	23.10	0.84	-0.49	-2.67
Ruwiki	16	4	70.00	39.00	14.30	68.00	42.10	16.60	1.14	-0.17	-1.29
Ruwiki	32	1	77.70	84.60	28.80	77.70	81.80	28.60	0.00	0.15	0.10
Ruwiki	32	2	69.20	53.50	18.50	67.20	53.10	18.80	1.13	0.02	-0.18
Ruwiki	32	3	68.80	45.00	16.90	67.60	44.10	16.80	0.67	0.05	0.09
Ruwiki	32	4	68.80	41.10	16.30	68.30	34.20	13.20	0.29	0.39	1.74
Zhwiki	4	1	34.40	45.60	11.80	33.90	39.40	11.90	0.77	0.90	-0.04
Zhwiki	4	2	27.70	35.90	7.91	26.50	32.80	7.95	1.78	0.45	-0.06
Zhwiki	4	3	27.60	26.80	6.71	26.80	27.10	7.70	1.23	-0.04	-1.46
Zhwiki	4	4	27.70	24.70	6.85	27.20	18.00	4.71	0.88	0.98	3.13
Zhwiki	16	1	33.00	37.90	13.20	33.10	36.80	13.00	-0.22	0.17	0.37
Zhwiki	16	2	26.70	27.20	9.71	25.90	27.70	10.10	1.23	-0.08	-0.63
Zhwiki	16	3	27.20	20.50	7.49	26.60	17.40	6.17	0.79	0.46	1.93
Zhwiki	16	4	27.50	16.20	6.03	26.20	20.80	8.50	1.91	-0.68	-3.63
Zhwiki	32	1	32.60	35.20	12.40	32.90	34.70	12.20	-0.36	0.08	0.30
Zhwiki	32	2	26.40	26.40	9.92	25.70	27.10	10.40	0.98	-0.10	-0.77
Zhwiki	32	3	27.10	16.90	6.40	26.20	19.00	7.35	1.39	-0.30	-1.40
Zhwiki	32	4	26.90	18.00	7.29	26.50	13.30	5.08	0.55	0.70	3.25
								μ	**0.83**	0.05	-0.53
								σ	0.60	0.49	1.92

Table D.96: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=\X, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	25.30	473.00	97.90	23.90	373.00	98.00	0.95	6.98	-0.10
Arwiki	4	2	17.90	337.00	63.10	17.40	284.00	62.00	0.38	3.70	0.78
Arwiki	4	3	18.40	252.00	48.50	17.00	249.00	73.00	0.91	0.17	-17.10
Arwiki	4	4	18.80	209.00	42.90	17.80	173.00	47.80	0.66	2.55	-3.37
Arwiki	16	1	23.90	374.00	98.30	23.50	330.00	97.80	0.33	3.03	0.32
Arwiki	16	2	17.60	217.00	50.50	17.50	188.00	49.50	0.06	2.07	0.69
Arwiki	16	3	17.30	176.00	56.80	16.90	194.00	67.80	0.26	-1.25	-7.69
Arwiki	16	4	17.80	136.00	42.40	17.90	115.00	36.00	-0.04	1.47	4.48
Arwiki	32	1	23.80	348.00	98.20	23.50	323.00	97.80	0.23	1.73	0.32
Arwiki	32	2	17.20	197.00	57.50	17.20	196.00	56.80	-0.00	0.10	0.45
Arwiki	32	3	17.70	141.00	42.10	17.90	136.00	37.10	-0.12	0.36	3.43
Arwiki	32	4	17.70	123.00	42.50	17.80	118.00	37.60	-0.11	0.33	3.40
Enwiki	4	1	17.30	288.00	60.10	16.50	241.00	60.10	0.82	4.72	0.04
Enwiki	4	2	11.80	238.00	44.90	11.30	204.00	49.60	0.55	3.44	-4.75
Enwiki	4	3	12.30	158.00	29.40	12.10	136.00	26.00	0.25	2.25	3.41
Enwiki	4	4	12.50	147.00	28.10	11.40	135.00	44.40	1.09	1.19	-16.40
Enwiki	16	1	16.30	239.00	60.30	16.10	213.00	60.20	0.22	2.70	0.11
Enwiki	16	2	11.20	169.00	49.20	11.10	163.00	49.60	0.08	0.63	-0.45
Enwiki	16	3	11.90	108.00	27.10	11.90	97.90	25.90	-0.00	1.05	1.17
Enwiki	16	4	12.00	90.10	26.20	11.20	108.00	44.30	0.77	-1.85	-18.20
Enwiki	32	1	16.20	218.00	60.20	16.10	220.00	60.00	0.12	-0.17	0.18
Enwiki	32	2	11.10	147.00	47.70	11.10	153.00	49.60	0.07	-0.63	-1.94
Enwiki	32	3	11.70	98.10	29.90	11.90	96.50	25.90	-0.16	0.16	3.96
Enwiki	32	4	11.70	84.30	29.10	11.20	107.00	44.30	0.48	-2.33	-15.40
Ruwiki	4	1	30.00	492.00	112.00	28.60	474.00	112.00	0.76	1.02	-0.17
Ruwiki	4	2	21.90	416.00	64.80	20.50	366.00	86.40	0.78	2.83	-12.20
Ruwiki	4	3	22.20	308.00	57.30	21.20	273.00	66.60	0.59	2.01	-5.29
Ruwiki	4	4	22.60	275.00	53.80	21.30	221.00	63.40	0.74	3.08	-5.43
Ruwiki	16	1	28.30	425.00	112.00	27.90	383.00	112.00	0.25	2.39	0.10
Ruwiki	16	2	21.00	273.00	67.00	20.90	235.00	65.60	0.07	2.14	0.75
Ruwiki	16	3	20.90	219.00	68.00	20.50	218.00	77.30	0.21	0.08	-5.26
Ruwiki	16	4	21.50	168.00	49.90	21.20	160.00	53.30	0.16	0.48	-1.96
Ruwiki	32	1	28.10	440.00	112.00	27.80	372.00	112.00	0.16	3.86	0.03
Ruwiki	32	2	20.80	239.00	68.90	20.70	239.00	68.60	0.01	-0.03	0.16
Ruwiki	32	3	21.10	184.00	59.20	21.10	180.00	57.30	-0.03	0.22	1.04
Ruwiki	32	4	21.20	149.00	52.90	21.50	144.00	43.70	-0.20	0.28	5.18
Zhwiki	4	1	13.20	216.00	47.30	12.70	183.00	47.20	0.81	4.77	0.13
Zhwiki	4	2	8.59	164.00	30.10	8.33	143.00	30.60	0.39	3.12	-0.74
Zhwiki	4	3	8.76	125.00	25.00	8.38	112.00	28.30	0.56	1.99	-4.84
Zhwiki	4	4	8.85	109.00	24.60	8.74	76.00	16.60	0.15	4.90	11.70
Zhwiki	16	1	12.60	178.00	47.50	12.40	168.00	47.30	0.27	1.42	0.27
Zhwiki	16	2	8.10	119.00	34.10	8.01	113.00	34.90	0.13	0.95	-1.16
Zhwiki	16	3	8.44	82.40	22.50	8.50	73.50	20.30	-0.09	1.31	3.20
Zhwiki	16	4	8.52	73.90	22.10	8.23	74.80	28.20	0.42	-0.12	-8.99
Zhwiki	32	1	12.50	166.00	47.50	12.40	160.00	47.30	0.13	0.86	0.31
Zhwiki	32	2	7.99	110.00	35.90	7.93	110.00	37.20	0.08	-0.10	-1.95
Zhwiki	32	3	8.35	75.10	24.20	8.32	78.10	25.40	0.04	-0.44	-1.82
Zhwiki	32	4	8.40	61.70	22.00	8.57	54.00	17.00	-0.25	1.12	7.38
								μ	**0.29**	1.47	-1.71
								σ	0.34	1.82	6.02

Table D.97: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=\X, Colours=never

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	21.20	93.40	77.90	20.30	92.20	77.90	0.63	0.08	0.01
Arwiki	4	2	15.80	113.00	49.70	15.20	97.20	49.70	0.41	1.08	0.02
Arwiki	4	3	17.20	80.10	31.00	14.40	96.20	59.20	1.96	-1.12	-19.60
Arwiki	4	4	16.90	75.40	35.90	16.00	67.90	38.60	0.62	0.52	-1.88
Arwiki	16	1	19.90	87.10	77.90	19.80	85.10	77.90	0.11	0.14	-0.01
Arwiki	16	2	15.50	74.20	39.60	15.30	57.90	39.60	0.14	1.14	0.01
Arwiki	16	3	15.70	60.90	35.10	14.10	68.80	55.00	1.16	-0.56	-13.90
Arwiki	16	4	15.90	53.90	33.40	16.00	39.50	29.10	-0.09	1.01	2.97
Arwiki	32	1	19.70	84.60	77.90	19.70	84.40	78.00	-0.01	0.02	-0.01
Arwiki	32	2	14.70	55.50	45.50	14.90	57.70	45.50	-0.09	-0.15	0.01
Arwiki	32	3	14.00	61.60	54.10	16.00	38.80	29.90	-1.43	1.60	16.90
Arwiki	32	4	15.20	43.80	37.90	15.90	37.40	30.30	-0.49	0.45	5.29
Enwiki	4	1	15.00	57.00	47.40	14.40	58.40	47.40	0.64	-0.14	-0.05
Enwiki	4	2	9.67	83.30	39.70	9.27	72.70	39.80	0.41	1.06	-0.07
Enwiki	4	3	10.70	59.70	26.10	10.70	46.10	20.70	-0.04	1.37	5.37
Enwiki	4	4	11.10	47.60	20.50	9.52	56.40	35.70	1.61	-0.89	-15.30
Enwiki	16	1	14.10	52.80	47.40	13.90	56.30	47.40	0.12	-0.35	-0.04
Enwiki	16	2	8.89	63.60	39.70	8.80	55.80	39.80	0.09	0.78	-0.12
Enwiki	16	3	9.84	43.80	26.30	10.30	32.00	20.80	-0.42	1.19	5.53
Enwiki	16	4	10.40	34.00	20.00	9.05	45.70	35.70	1.35	-1.18	-15.80
Enwiki	32	1	13.90	52.20	47.40	13.90	57.90	47.50	-0.01	-0.58	-0.07
Enwiki	32	2	8.70	47.80	39.70	8.79	54.50	39.80	-0.09	-0.68	-0.08
Enwiki	32	3	9.28	37.30	31.50	10.20	30.40	20.80	-0.98	0.69	10.80
Enwiki	32	4	10.10	24.30	20.20	9.03	44.60	35.80	1.13	-2.04	-15.70
Ruwiki	4	1	24.90	106.00	88.40	23.90	112.00	88.40	0.59	-0.34	-0.03
Ruwiki	4	2	17.90	151.00	69.40	17.20	129.00	69.40	0.42	1.28	-0.01
Ruwiki	4	3	20.10	103.00	39.90	18.30	100.00	53.80	1.03	0.17	-7.83
Ruwiki	4	4	19.80	94.50	43.70	18.50	87.90	51.40	0.75	0.37	-4.35
Ruwiki	16	1	23.20	95.60	88.40	23.10	97.70	88.40	0.09	-0.12	-0.01
Ruwiki	16	2	18.00	98.90	52.50	17.70	74.50	52.50	0.17	1.38	-0.01
Ruwiki	16	3	18.80	71.60	38.80	16.80	79.70	62.50	1.13	-0.46	-13.40
Ruwiki	16	4	18.90	64.50	39.00	18.30	56.00	43.20	0.32	0.48	-2.39
Ruwiki	32	1	22.90	94.10	88.40	22.90	94.80	88.40	-0.02	-0.04	-0.02
Ruwiki	32	2	17.40	67.40	55.00	17.50	69.60	55.00	-0.10	-0.13	0.01
Ruwiki	32	3	17.20	64.90	55.70	18.10	57.00	46.20	-0.53	0.45	5.38
Ruwiki	32	4	17.90	52.80	45.80	18.90	43.80	35.30	-0.56	0.51	5.92
Zhwiki	4	1	10.80	44.60	37.50	10.50	46.00	37.50	0.56	-0.20	-0.01
Zhwiki	4	2	7.70	56.90	24.60	7.40	48.00	24.60	0.44	1.30	-0.01
Zhwiki	4	3	8.11	42.00	16.30	7.34	41.30	22.70	1.13	0.11	-9.40
Zhwiki	4	4	7.96	38.70	18.10	8.07	27.70	13.30	-0.16	1.61	7.09
Zhwiki	16	1	10.10	41.10	37.60	10.20	42.70	37.60	-0.02	-0.23	0.07
Zhwiki	16	2	6.80	43.80	28.00	6.74	37.40	28.00	0.09	0.93	-0.04
Zhwiki	16	3	6.81	37.90	26.20	7.54	23.30	16.20	-1.08	2.13	14.70
Zhwiki	16	4	7.61	25.60	16.20	7.03	28.20	22.70	0.85	-0.38	-9.56
Zhwiki	32	1	10.00	40.30	37.60	10.10	42.00	37.60	-0.13	-0.25	-0.02
Zhwiki	32	2	6.49	35.20	29.90	6.55	36.40	29.80	-0.09	-0.17	0.05
Zhwiki	32	3	7.51	19.40	15.60	7.21	24.90	20.40	0.44	-0.81	-7.09
Zhwiki	32	4	6.88	25.90	22.70	7.71	17.30	13.60	-1.22	1.26	13.40
								μ	**0.23**	0.26	-0.91
								σ	0.69	0.85	7.61

Table D.98: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=\X, Colours=never

D.4.8 Case study: `icgrep -colours=always`

Arguments: @ -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	12.30	22.40	4.87	7.62	11.90	5.82	3.27	0.73	-0.66
Arwiki	4	2	6.99	35.90	2.85	4.46	35.10	2.56	1.77	0.05	0.20
Arwiki	4	3	4.80	31.50	1.96	3.10	26.40	1.72	1.19	0.36	0.16
Arwiki	4	4	4.24	22.90	1.21	2.47	21.70	1.14	1.23	0.08	0.05
Arwiki	16	1	9.23	24.80	6.54	7.32	16.80	7.79	1.33	0.56	-0.87
Arwiki	16	2	5.08	20.40	2.94	4.00	19.50	3.22	0.75	0.06	-0.20
Arwiki	16	3	3.71	13.20	1.90	2.56	10.40	1.43	0.80	0.19	0.32
Arwiki	16	4	3.36	10.30	1.17	2.13	11.10	1.44	0.86	-0.06	-0.19
Arwiki	32	1	8.61	27.60	6.94	7.20	12.90	5.54	0.98	1.03	0.97
Arwiki	32	2	4.68	17.20	3.09	3.82	15.20	2.97	0.60	0.14	0.08
Arwiki	32	3	3.45	11.50	1.98	2.48	8.07	1.42	0.68	0.24	0.39
Arwiki	32	4	3.07	8.38	1.31	1.91	5.98	.86	0.81	0.17	0.31
Enwiki	4	1	9.09	14.30	3.54	5.23	5.74	2.50	3.89	0.86	1.05
Enwiki	4	2	4.75	25.70	1.96	3.06	24.60	1.83	1.70	0.10	0.13
Enwiki	4	3	3.49	18.70	1.25	2.10	18.70	1.17	1.39	-0.00	0.08
Enwiki	4	4	2.94	14.70	.80	1.60	14.90	.80	1.34	-0.02	-0.00
Enwiki	16	1	6.42	15.00	4.46	5.03	10.40	4.97	1.40	0.47	-0.52
Enwiki	16	2	3.67	11.90	2.00	2.74	13.30	2.19	0.94	-0.14	-0.19
Enwiki	16	3	2.53	8.66	1.34	1.75	6.97	.96	0.79	0.17	0.38
Enwiki	16	4	2.22	6.54	.84	1.45	7.40	.98	0.77	-0.09	-0.15
Enwiki	32	1	5.96	17.70	4.76	4.97	8.46	3.77	1.00	0.93	1.00
Enwiki	32	2	3.26	10.70	2.14	2.62	9.89	2.05	0.64	0.08	0.09
Enwiki	32	3	2.35	7.67	1.42	1.70	5.39	.93	0.65	0.23	0.50
Enwiki	32	4	2.11	5.32	.91	1.31	3.98	.58	0.81	0.14	0.33
Ruwiki	4	1	15.20	28.60	5.95	9.42	14.00	6.66	3.28	0.83	-0.40
Ruwiki	4	2	8.47	49.10	3.68	5.50	42.80	3.23	1.68	0.35	0.25
Ruwiki	4	3	6.02	37.60	2.36	3.86	32.90	2.11	1.22	0.27	0.14
Ruwiki	4	4	5.28	29.20	1.47	2.99	26.90	1.39	1.29	0.13	0.05
Ruwiki	16	1	11.40	30.40	8.08	9.04	20.30	9.34	1.32	0.58	-0.72
Ruwiki	16	2	6.34	25.70	3.60	5.00	25.30	3.84	0.76	0.02	-0.14
Ruwiki	16	3	4.60	16.80	2.32	3.21	13.30	1.71	0.79	0.20	0.35
Ruwiki	16	4	4.11	13.10	1.39	2.66	14.50	1.75	0.82	-0.08	-0.20
Ruwiki	32	1	10.70	32.90	8.49	8.89	15.70	6.66	1.01	0.97	1.04
Ruwiki	32	2	5.95	21.60	3.93	4.78	20.10	3.68	0.66	0.08	0.14
Ruwiki	32	3	4.48	14.00	2.42	3.09	10.20	1.65	0.79	0.21	0.44
Ruwiki	32	4	3.63	10.20	1.59	2.39	8.05	1.06	0.70	0.12	0.30
Zhwiki	4	1	5.98	11.00	2.35	3.66	5.30	2.43	3.40	0.83	-0.12
Zhwiki	4	2	3.25	19.80	1.43	2.14	16.90	1.27	1.62	0.43	0.23
Zhwiki	4	3	2.38	14.70	.90	1.51	12.70	.82	1.28	0.28	0.12
Zhwiki	4	4	2.08	11.00	.59	1.17	10.50	.57	1.35	0.07	0.03
Zhwiki	16	1				3.52	8.13	3.63			
Zhwiki	16	2	2.45	10.20	1.44	1.94	9.50	1.51	0.75	0.10	-0.10
Zhwiki	16	3	1.85	6.43	.88	1.26	5.18	.67	0.87	0.18	0.31
Zhwiki	16	4	1.56	4.99	.56	1.04	5.45	.69	0.75	-0.07	-0.20
Zhwiki	32	1				3.47	6.31	2.53			
Zhwiki	32	2	2.32	8.27	1.49	1.84	7.37	1.44	0.69	0.13	0.07
Zhwiki	32	3	1.74	5.60	.95	1.22	4.03	.65	0.77	0.23	0.45
Zhwiki	32	4	1.41	4.28	.71	.95	3.05	.40	0.68	0.18	0.45
								μ	**1.22**	0.27	0.12
								σ	0.76	0.30	0.41

Table D.99: `icgrep` PAPI comparison between 2020 and `CURR` on `SSE-4.2` with arguments Expression=@, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1				2.63	54.60	23.00			
Arwiki	4	2	3.33	234.00	12.00	2.03	152.00	11.40	0.91	5.76	0.45
Arwiki	4	3	2.34	183.00	8.00	1.50	121.00	7.63	0.59	4.32	0.26
Arwiki	4	4	1.97	149.00	6.09	1.21	100.00	5.76	0.53	3.38	0.23
Arwiki	16	1				2.45	57.40	23.50			
Arwiki	16	2	1.98	110.00	12.00	1.50	75.30	11.80	0.34	2.42	0.16
Arwiki	16	3	1.44	79.40	8.26	.98	51.10	7.92	0.32	1.97	0.23
Arwiki	16	4	1.22	65.50	6.24	.87	46.00	6.00	0.24	1.36	0.16
Arwiki	32	1				2.44	66.10	23.50			
Arwiki	32	2	1.72	93.80	12.10	1.38	58.70	11.80	0.23	2.45	0.21
Arwiki	32	3	1.22	66.40	8.26	.93	41.90	7.95	0.21	1.71	0.21
Arwiki	32	4	1.01	51.60	6.24	.76	33.00	5.95	0.18	1.29	0.20
Enwiki	4	1	4.48	78.50	16.80	1.80	30.50	15.90	2.70	4.83	0.87
Enwiki	4	2	2.39	142.00	8.24	1.38	106.00	7.82	1.02	3.63	0.43
Enwiki	4	3	1.75	107.00	5.56	1.02	85.50	5.24	0.73	2.17	0.32
Enwiki	4	4	1.37	98.70	4.16	.82	71.20	3.96	0.56	2.77	0.20
Enwiki	16	1	2.57	66.70	16.40	1.67	33.50	16.20	0.91	3.35	0.18
Enwiki	16	2	1.42	67.50	8.31	1.01	51.50	8.06	0.41	1.61	0.25
Enwiki	16	3	.96	50.20	5.58	.66	34.70	5.41	0.30	1.56	0.18
Enwiki	16	4	.81	41.80	4.22	.58	31.40	4.09	0.22	1.05	0.13
Enwiki	32	1	2.21	78.70	16.20	1.66	41.50	16.20	0.55	3.75	0.07
Enwiki	32	2	1.18	60.00	8.21	.94	39.60	8.10	0.24	2.06	0.11
Enwiki	32	3	.83	41.80	5.59	.64	28.00	5.43	0.19	1.39	0.15
Enwiki	32	4	.68	33.00	4.19	.53	22.30	4.07	0.15	1.08	0.13
Ruwiki	4	1	7.30	151.00	30.10	3.28	60.80	28.30	2.28	5.09	1.03
Ruwiki	4	2	4.13	295.00	14.90	2.54	186.00	14.10	0.90	6.19	0.47
Ruwiki	4	3	2.92	230.00	9.87	1.86	149.00	9.39	0.60	4.59	0.27
Ruwiki	4	4	2.32	192.00	7.46	1.50	125.00	7.18	0.46	3.79	0.16
Ruwiki	16	1	4.56	134.00	29.70	3.04	67.60	29.00	0.86	3.77	0.38
Ruwiki	16	2	2.48	136.00	15.00	1.88	94.00	14.50	0.34	2.38	0.26
Ruwiki	16	3	1.70	101.00	10.10	1.23	64.30	9.77	0.27	2.10	0.21
Ruwiki	16	4	1.47	83.20	7.65	1.09	57.50	7.44	0.21	1.46	0.12
Ruwiki	32	1	4.01	151.00	29.60	3.03	79.00	29.10	0.56	4.06	0.30
Ruwiki	32	2	2.10	113.00	14.90	1.75	74.60	14.60	0.20	2.18	0.16
Ruwiki	32	3	1.47	80.10	10.20	1.17	53.10	9.81	0.17	1.53	0.20
Ruwiki	32	4	1.20	64.20	7.67	.96	42.50	7.41	0.14	1.23	0.15
Zhwiki	4	1	2.88	58.10	11.70	1.27	24.10	11.00	2.37	5.00	1.13
Zhwiki	4	2	1.62	114.00	5.86	.98	72.90	5.44	0.94	6.05	0.61
Zhwiki	4	3	1.18	83.60	3.95	.73	58.30	3.66	0.67	3.71	0.43
Zhwiki	4	4	1.01	67.10	3.04	.59	48.80	2.80	0.62	2.68	0.34
Zhwiki	16	1	1.78	53.00	11.70	1.18	27.40	11.20	0.87	3.76	0.64
Zhwiki	16	2	.97	54.00	5.91	.73	36.60	5.65	0.36	2.56	0.39
Zhwiki	16	3	.72	38.20	4.10	.49	25.10	3.83	0.34	1.93	0.38
Zhwiki	16	4	.63	29.90	3.14	.44	22.10	2.91	0.29	1.16	0.34
Zhwiki	32	1				1.18	32.00	11.30			
Zhwiki	32	2	.85	44.80	5.95	.69	28.90	5.70	0.24	2.33	0.36
Zhwiki	32	3	.65	32.00	4.15	.48	20.60	3.87	0.25	1.67	0.41
Zhwiki	32	4	.55	25.60	3.15	.40	16.20	2.89	0.21	1.38	0.37
									0.58	2.83	0.32
									0.57	1.43	0.23

Table D.100: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=@, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	4.60	23.50	22.70	1.72	23.50	22.50	2.01	-0.00	0.08
Arwiki	4	2	3.38	94.70	11.50	1.96	77.20	11.40	0.99	1.22	0.07
Arwiki	4	3	2.57	79.40	7.67	1.61	62.20	7.60	0.67	1.20	0.05
Arwiki	4	4	2.17	68.00	5.74	1.35	51.50	5.70	0.57	1.15	0.03
Arwiki	16	1	2.55	23.50	22.70	1.55	23.40	22.60	0.70	0.01	0.08
Arwiki	16	2	1.61	48.30	11.50	1.15	36.50	11.40	0.32	0.82	0.09
Arwiki	16	3	1.09	32.90	7.69	.87	24.20	7.61	0.15	0.61	0.05
Arwiki	16	4	.97	30.10	5.77	.75	19.70	5.72	0.15	0.72	0.04
Arwiki	32	1	2.15	23.50	22.70	1.53	23.50	22.60	0.43	0.01	0.06
Arwiki	32	2	1.28	35.00	11.60	1.03	25.20	11.40	0.17	0.68	0.12
Arwiki	32	3	.85	23.80	7.72	.75	18.20	7.64	0.07	0.39	0.06
Arwiki	32	4	.72	19.00	5.80	.72	14.60	5.74	-0.01	0.30	0.05
Enwiki	4	1	3.49	15.80	15.60	1.20	15.90	15.60	2.30	-0.01	0.02
Enwiki	4	2	2.37	64.50	7.87	1.38	54.40	7.82	1.00	1.02	0.05
Enwiki	4	3	1.79	55.20	5.24	1.13	44.60	5.22	0.67	1.07	0.03
Enwiki	4	4	1.50	46.40	3.93	.95	36.50	3.91	0.55	0.99	0.02
Enwiki	16	1	1.81	15.90	15.60	1.07	15.90	15.60	0.75	0.00	0.03
Enwiki	16	2	1.10	29.90	7.88	.79	25.10	7.84	0.31	0.48	0.05
Enwiki	16	3	.74	21.00	5.25	.60	16.60	5.23	0.14	0.45	0.03
Enwiki	16	4	.66	18.00	3.95	.52	13.40	3.93	0.14	0.46	0.02
Enwiki	32	1	1.48	15.90	15.60	1.06	15.90	15.60	0.43	-0.00	0.02
Enwiki	32	2	.88	20.60	7.94	.73	17.20	7.85	0.15	0.35	0.10
Enwiki	32	3	.58	13.90	5.29	.54	12.40	5.24	0.05	0.14	0.05
Enwiki	32	4	.50	11.10	3.98	.54	10.00	3.95	-0.04	0.11	0.03
Ruwiki	4	1	5.68	29.30	28.00	2.13	29.30	27.80	2.01	0.01	0.11
Ruwiki	4	2	4.15	118.00	14.30	2.41	95.40	14.00	0.98	1.28	0.13
Ruwiki	4	3	3.20	98.80	9.48	1.99	76.90	9.38	0.69	1.24	0.06
Ruwiki	4	4	2.66	84.50	7.13	1.66	64.00	7.06	0.57	1.17	0.04
Ruwiki	16	1	3.19	29.30	28.10	1.92	29.10	27.80	0.72	0.01	0.14
Ruwiki	16	2	2.01	59.80	14.20	1.43	45.30	14.10	0.33	0.82	0.10
Ruwiki	16	3	1.37	44.20	9.54	1.07	30.20	9.42	0.17	0.79	0.07
Ruwiki	16	4	1.24	37.70	7.15	.91	24.40	7.07	0.19	0.75	0.05
Ruwiki	32	1	2.67	29.40	28.10	1.89	29.30	27.90	0.44	0.01	0.10
Ruwiki	32	2	1.63	41.50	14.30	1.27	31.20	14.10	0.20	0.58	0.12
Ruwiki	32	3	1.06	28.10	9.61	.91	22.60	9.45	0.09	0.31	0.09
Ruwiki	32	4	.89	22.40	7.21	.87	18.30	7.13	0.01	0.24	0.05
Zhwiki	4	1	2.23	11.70	10.90	.83	11.60	10.80	2.05	0.02	0.22
Zhwiki	4	2	1.63	46.20	5.57	.94	37.00	5.47	1.00	1.34	0.15
Zhwiki	4	3	1.25	38.90	3.73	.78	29.80	3.66	0.69	1.32	0.10
Zhwiki	4	4	1.05	32.70	2.80	.66	24.80	2.77	0.57	1.15	0.05
Zhwiki	16	1	1.24	11.70	11.00	.76	11.60	10.80	0.71	0.02	0.24
Zhwiki	16	2	.80	23.60	5.61	.57	17.60	5.48	0.34	0.89	0.19
Zhwiki	16	3	.56	17.10	3.76	.45	11.70	3.67	0.16	0.79	0.12
Zhwiki	16	4	.50	14.60	2.83	.40	9.50	2.77	0.16	0.75	0.09
Zhwiki	32	1	1.05	11.80	11.00	.75	11.60	10.90	0.43	0.02	0.18
Zhwiki	32	2	.64	16.40	5.65	.53	12.20	5.52	0.17	0.61	0.20
Zhwiki	32	3	.45	11.30	3.80	.41	8.82	3.70	0.05	0.36	0.14
Zhwiki	32	4	.39	9.04	2.86	.41	7.11	2.79	-0.02	0.28	0.11
								μ	**0.53**	0.56	0.08
								σ	0.56	0.45	0.06

Table D.101: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=@, Colours=always

Arguments: Date -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	14.30	32.90	2.77	9.61	18.00	4.92	3.28	1.04	-1.50
Arwiki	4	2	8.23	45.30	1.83	5.90	37.80	2.07	1.63	0.52	-0.17
Arwiki	4	3	5.73	41.00	1.59	4.36	28.90	1.49	0.96	0.85	0.07
Arwiki	4	4	5.25	30.10	1.08	3.57	23.90	1.05	1.17	0.43	0.02
Arwiki	16	1	11.10	30.50	4.02	9.22	17.50	4.55	1.29	0.91	-0.37
Arwiki	16	2	5.95	24.80	2.03	5.10	16.80	1.57	0.59	0.56	0.32
Arwiki	16	3	4.38	17.30	1.47	3.60	12.50	1.33	0.54	0.34	0.10
Arwiki	16	4	4.22	12.50	.91	2.97	10.40	.76	0.87	0.15	0.11
Arwiki	32	1	10.40	29.30	4.43	9.13	21.10	6.07	0.86	0.57	-1.14
Arwiki	32	2	5.72	18.90	2.19	4.93	15.40	2.50	0.55	0.25	-0.22
Arwiki	32	3	4.25	13.40	1.58	3.42	11.50	2.20	0.57	0.13	-0.43
Arwiki	32	4	3.79	9.92	1.12	2.76	7.48	.91	0.72	0.17	0.15
Enwiki	4	1	9.39	9.91	1.63	5.65	3.82	1.84	3.76	0.61	-0.22
Enwiki	4	2	5.08	18.00	.90	3.29	23.60	1.22	1.81	-0.57	-0.33
Enwiki	4	3	3.61	17.80	.58	2.24	17.80	.79	1.38	0.00	-0.21
Enwiki	4	4	3.04	13.70	.41	1.73	14.70	.51	1.32	-0.10	-0.10
Enwiki	16	1	6.73	9.60	1.75	5.43	4.70	2.58	1.31	0.49	-0.83
Enwiki	16	2	3.74	9.51	.79	2.88	8.94	.69	0.86	0.06	0.10
Enwiki	16	3	2.67	6.87	.52	1.87	5.73	.59	0.81	0.12	-0.07
Enwiki	16	4	2.40	4.96	.26	1.54	5.61	.24	0.86	-0.07	0.02
Enwiki	32	1	6.21	11.20	1.83	5.39	8.43	3.74	0.82	0.28	-1.93
Enwiki	32	2	3.38	7.68	.78	2.80	7.55	1.13	0.58	0.01	-0.35
Enwiki	32	3	2.55	5.62	.54	1.84	5.13	.89	0.71	0.05	-0.36
Enwiki	32	4	2.18	3.70	.30	1.41	3.15	.29	0.77	0.06	0.02
Ruwiki	4	1	17.90	44.60	3.91	12.10	23.60	5.91	3.29	1.19	-1.13
Ruwiki	4	2	10.80	53.50	2.62	7.44	47.40	2.63	1.91	0.35	-0.01
Ruwiki	4	3	7.92	45.90	1.80	5.58	36.90	1.88	1.33	0.51	-0.05
Ruwiki	4	4	7.03	34.80	1.34	4.78	30.10	1.37	1.27	0.27	-0.02
Ruwiki	16	1	13.90	40.50	5.17	11.60	22.60	5.45	1.31	1.01	-0.15
Ruwiki	16	2	7.81	29.90	2.34	6.47	22.00	1.95	0.76	0.45	0.22
Ruwiki	16	3	6.08	21.50	1.68	4.60	16.70	1.80	0.84	0.27	-0.07
Ruwiki	16	4	5.30	17.00	1.24	3.80	13.60	.99	0.85	0.19	0.14
Ruwiki	32	1	13.10	37.40	5.52	11.50	27.50	7.48	0.91	0.57	-1.11
Ruwiki	32	2	7.32	23.90	2.66	6.17	20.80	3.41	0.65	0.18	-0.43
Ruwiki	32	3	5.68	17.00	1.93	4.40	15.00	2.43	0.72	0.11	-0.28
Ruwiki	32	4	4.81	13.20	1.47	3.49	10.90	1.37	0.75	0.13	0.06
Zhwiki	4	1	7.25	18.30	1.35	4.93	10.40	2.31	3.42	1.16	-1.40
Zhwiki	4	2	4.23	24.40	1.01	3.06	19.20	1.10	1.71	0.77	-0.13
Zhwiki	4	3	3.35	18.50	.73	2.32	15.00	.82	1.52	0.52	-0.14
Zhwiki	4	4	2.85	15.20	.56	1.99	12.00	.60	1.26	0.47	-0.06
Zhwiki	16	1				4.70	10.20	2.17			
Zhwiki	16	2	3.21	12.50	1.03	2.64	8.99	.83	0.83	0.52	0.29
Zhwiki	16	3	2.44	9.26	.79	1.89	7.11	.82	0.80	0.32	-0.05
Zhwiki	16	4	2.11	7.09	.57	1.56	5.66	.51	0.81	0.21	0.08
Zhwiki	32	1				4.67	11.90	2.92			
Zhwiki	32	2	3.03	10.10	1.30	2.53	8.47	1.42	0.73	0.24	-0.18
Zhwiki	32	3	2.37	7.10	.86	1.81	6.28	1.09	0.82	0.12	-0.34
Zhwiki	32	4	1.97	5.41	.64	1.46	4.34	.58	0.75	0.16	0.09
								μ	**1.20**	0.36	-0.26
								σ	0.77	0.35	0.49

Table D.102: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=Date, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	6.64	207.00	32.50	3.37	108.00	33.40	2.28	6.91	-0.66
Arwiki	4	2	3.86	282.00	18.40	2.57	179.00	17.00	0.90	7.16	1.00
Arwiki	4	3	2.86	223.00	13.00	1.97	139.00	11.10	0.62	5.85	1.35
Arwiki	4	4	2.38	185.00	10.30	1.62	114.00	8.80	0.53	5.00	1.03
Arwiki	16	1	4.38	190.00	37.50	3.20	112.00	33.40	0.82	5.49	2.86
Arwiki	16	2	2.45	149.00	20.60	1.96	98.80	17.00	0.34	3.47	2.53
Arwiki	16	3	1.81	108.00	14.40	1.39	68.70	12.30	0.30	2.78	1.41
Arwiki	16	4	1.55	86.30	11.20	1.20	55.50	8.93	0.24	2.15	1.56
Arwiki	32	1	3.87	204.00	39.70	3.18	117.00	33.50	0.49	6.08	4.31
Arwiki	32	2	2.11	130.00	21.60	1.83	80.20	17.30	0.20	3.49	3.03
Arwiki	32	3	1.52	92.30	15.50	1.32	57.50	12.70	0.14	2.43	1.92
Arwiki	32	4	1.36	71.70	11.60	1.12	44.60	9.72	0.17	1.89	1.33
Enwiki	4	1	4.61	93.70	21.30	2.02	35.20	21.20	2.61	5.90	0.06
Enwiki	4	2	2.47	145.00	10.80	1.48	109.00	10.60	1.00	3.62	0.19
Enwiki	4	3	1.80	106.00	7.02	1.09	87.40	6.88	0.71	1.88	0.14
Enwiki	4	4	1.37	101.00	5.22	.86	70.90	5.22	0.51	3.05	-0.01
Enwiki	16	1	2.71	71.20	21.30	1.91	37.80	21.30	0.80	3.36	-0.03
Enwiki	16	2	1.44	72.80	10.90	1.13	54.40	10.90	0.31	1.85	0.02
Enwiki	16	3	.95	52.80	7.25	.74	35.60	7.16	0.21	1.74	0.10
Enwiki	16	4	.81	42.40	5.33	.64	30.50	5.27	0.16	1.20	0.06
Enwiki	32	1	2.32	83.50	21.40	1.90	46.00	21.40	0.43	3.77	0.04
Enwiki	32	2	1.25	61.10	10.90	1.06	40.50	10.90	0.19	2.08	0.06
Enwiki	32	3	.87	43.80	7.33	.72	28.80	7.23	0.16	1.51	0.10
Enwiki	32	4	.73	34.30	5.44	.59	22.80	5.37	0.14	1.15	0.08
Ruwiki	4	1	8.34	282.00	40.80	4.25	134.00	41.30	2.32	8.40	-0.28
Ruwiki	4	2	4.86	357.00	23.10	3.20	226.00	21.50	0.94	7.42	0.89
Ruwiki	4	3	3.64	280.00	16.40	2.47	176.00	14.00	0.67	5.89	1.35
Ruwiki	4	4	2.99	231.00	12.60	2.04	143.00	11.30	0.53	5.02	0.72
Ruwiki	16	1	5.49	245.00	47.00	4.02	142.00	41.10	0.83	5.82	3.32
Ruwiki	16	2	3.07	193.00	26.40	2.45	126.00	21.80	0.35	3.81	2.62
Ruwiki	16	3	2.20	141.00	18.30	1.78	88.40	14.80	0.24	2.98	2.01
Ruwiki	16	4	1.88	114.00	14.20	1.51	70.70	11.10	0.21	2.46	1.75
Ruwiki	32	1	4.90	246.00	49.70	3.99	153.00	41.10	0.52	5.25	4.87
Ruwiki	32	2	2.65	164.00	27.40	2.29	103.00	21.70	0.21	3.43	3.23
Ruwiki	32	3	1.99	115.00	19.30	1.69	74.20	15.20	0.17	2.32	2.34
Ruwiki	32	4	1.69	90.40	14.90	1.40	58.40	12.40	0.17	1.81	1.42
Zhwiki	4	1	3.37	114.00	16.10	1.71	58.00	16.20	2.43	8.18	-0.20
Zhwiki	4	2	2.00	141.00	9.31	1.29	90.40	8.36	1.04	7.40	1.39
Zhwiki	4	3	1.53	109.00	6.83	1.00	70.50	5.85	0.78	5.68	1.45
Zhwiki	4	4	1.32	90.00	5.45	.85	57.40	4.45	0.69	4.78	1.48
Zhwiki	16	1	2.22	108.00	19.10	1.62	62.20	16.20	0.88	6.76	4.24
Zhwiki	16	2	1.27	80.60	10.90	.99	51.50	8.34	0.40	4.27	3.77
Zhwiki	16	3	.96	58.10	7.73	.75	36.40	5.83	0.31	3.19	2.78
Zhwiki	16	4	.81	46.80	6.07	.63	28.90	4.57	0.26	2.62	2.20
Zhwiki	32	1				1.61	65.80	16.30			
Zhwiki	32	2	1.13	68.70	11.50	.92	43.80	9.00	0.30	3.65	3.63
Zhwiki	32	3	.84	48.70	8.12	.71	31.20	6.32	0.18	2.57	2.65
Zhwiki	32	4	.75	38.40	6.44	.60	24.60	5.38	0.21	2.02	1.55
								μ	**0.61**	4.03	1.52
								σ	0.61	2.00	1.39

Table D.103: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=Date, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	5.03	33.20	23.90	2.17	29.70	24.50	2.00	0.24	-0.40
Arwiki	4	2	3.84	105.00	13.80	2.38	86.10	13.00	1.02	1.31	0.57
Arwiki	4	3	3.02	88.80	9.70	2.02	68.20	8.74	0.70	1.44	0.67
Arwiki	4	4	2.73	78.00	7.73	1.75	56.80	6.82	0.69	1.48	0.64
Arwiki	16	1	2.98	35.70	26.30	2.00	30.90	24.60	0.69	0.33	1.19
Arwiki	16	2	2.00	58.80	15.20	1.53	42.00	13.00	0.33	1.17	1.59
Arwiki	16	3	1.53	41.60	10.50	1.21	29.20	8.96	0.23	0.86	1.07
Arwiki	16	4	1.37	36.70	8.05	1.07	24.90	7.05	0.20	0.83	0.70
Arwiki	32	1	2.56	37.50	27.50	1.99	31.40	24.70	0.40	0.42	1.99
Arwiki	32	2	1.65	43.90	15.60	1.40	30.80	13.20	0.18	0.92	1.67
Arwiki	32	3	1.24	31.00	10.70	1.05	23.00	9.71	0.13	0.56	0.72
Arwiki	32	4	1.10	24.80	8.31	.99	18.30	7.30	0.08	0.46	0.71
Enwiki	4	1	3.54	15.80	15.60	1.32	15.90	15.50	2.24	-0.01	0.01
Enwiki	4	2	2.41	65.10	7.84	1.43	56.80	7.78	0.99	0.84	0.06
Enwiki	4	3	1.81	54.90	5.22	1.14	45.70	5.21	0.68	0.93	0.01
Enwiki	4	4	1.52	46.80	3.92	.95	37.40	3.90	0.57	0.95	0.03
Enwiki	16	1	1.84	15.80	15.60	1.17	15.80	15.50	0.68	-0.00	0.01
Enwiki	16	2	1.12	30.80	7.88	.83	26.20	7.82	0.29	0.47	0.06
Enwiki	16	3	.75	21.80	5.24	.61	17.20	5.22	0.15	0.47	0.02
Enwiki	16	4	.68	18.90	3.95	.55	15.10	3.92	0.13	0.38	0.03
Enwiki	32	1	1.49	15.80	15.60	1.16	15.80	15.60	0.33	-0.01	0.01
Enwiki	32	2	.90	21.20	7.92	.77	17.70	7.84	0.12	0.35	0.09
Enwiki	32	3	.59	14.20	5.27	.54	12.80	5.23	0.05	0.14	0.04
Enwiki	32	4	.51	11.40	3.97	.50	10.40	3.94	0.01	0.10	0.03
Ruwiki	4	1	6.26	41.90	29.30	2.72	37.40	30.20	2.00	0.26	-0.53
Ruwiki	4	2	4.76	133.00	17.10	2.97	107.00	16.20	1.02	1.47	0.51
Ruwiki	4	3	3.90	112.00	12.70	2.51	85.00	11.00	0.78	1.55	0.97
Ruwiki	4	4	3.39	98.10	9.69	2.18	70.80	8.48	0.68	1.55	0.68
Ruwiki	16	1	3.74	44.80	32.60	2.51	39.00	30.30	0.69	0.33	1.32
Ruwiki	16	2	2.52	73.20	19.20	1.84	53.10	16.80	0.39	1.14	1.37
Ruwiki	16	3	1.93	56.00	13.40	1.47	37.40	11.80	0.26	1.05	0.93
Ruwiki	16	4	1.74	47.00	10.20	1.35	31.10	8.69	0.22	0.90	0.88
Ruwiki	32	1	3.23	47.20	34.30	2.49	39.90	30.50	0.42	0.41	2.19
Ruwiki	32	2	2.07	53.40	19.70	1.73	38.70	16.40	0.20	0.83	1.83
Ruwiki	32	3	1.54	38.00	13.80	1.30	29.00	12.10	0.14	0.51	0.92
Ruwiki	32	4	1.38	30.50	10.60	1.22	23.00	9.06	0.09	0.43	0.86
Zhwiki	4	1	2.53	17.60	11.80	1.10	15.10	11.90	2.10	0.37	-0.10
Zhwiki	4	2	1.96	53.40	7.10	1.21	42.20	6.30	1.10	1.66	1.19
Zhwiki	4	3	1.59	45.60	5.25	1.03	33.40	4.38	0.83	1.79	1.29
Zhwiki	4	4	1.42	39.50	4.08	.91	27.80	3.40	0.76	1.71	1.00
Zhwiki	16	1	1.52	18.90	13.10	1.03	16.00	12.00	0.73	0.43	1.75
Zhwiki	16	2	1.05	30.00	7.98	.77	21.20	6.77	0.41	1.29	1.77
Zhwiki	16	3	.82	22.70	5.61	.65	14.90	4.54	0.26	1.14	1.56
Zhwiki	16	4	.75	19.10	4.37	.56	12.90	3.93	0.28	0.92	0.64
Zhwiki	32	1	1.31	19.50	14.00	1.03	16.40	12.00	0.42	0.46	2.88
Zhwiki	32	2	.86	22.10	8.21	.71	16.10	6.99	0.21	0.88	1.78
Zhwiki	32	3	.68	15.80	5.76	.61	11.70	4.80	0.11	0.61	1.41
Zhwiki	32	4	.62	12.80	4.49	.57	9.44	3.86	0.07	0.49	0.93
								μ	**0.56**	0.77	0.82
								σ	0.54	0.49	0.74

Table D.104: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=Date, Colours=always

Arguments: Email -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	20.70	49.80	3.02	13.80	13.80	2.83	4.82	2.51	0.13
Arwiki	4	2	10.80	55.30	1.32	7.59	43.30	.98	2.22	0.84	0.23
Arwiki	4	3	7.68	40.50	.93	4.99	28.20	.72	1.88	0.85	0.15
Arwiki	4	4	6.71	32.30	.64	3.98	27.00	.45	1.91	0.37	0.13
Arwiki	16	1	15.90	36.00	2.07	13.10	14.50	2.41	1.91	1.50	-0.23
Arwiki	16	2	8.79	23.50	.95	6.81	18.40	.96	1.38	0.36	-0.00
Arwiki	16	3	6.17	16.50	.62	4.50	11.90	.68	1.17	0.32	-0.04
Arwiki	16	4	5.32	13.60	.39	3.65	10.70	.36	1.17	0.20	0.02
Arwiki	32	1	15.00	36.00	2.32	13.00	17.50	2.52	1.37	1.29	-0.14
Arwiki	32	2	8.21	20.00	1.20	6.71	15.00	1.07	1.04	0.35	0.09
Arwiki	32	3	5.60	13.50	.74	4.40	9.61	.81	0.84	0.27	-0.05
Arwiki	32	4	5.42	9.33	.42	3.39	7.64	.57	1.42	0.12	-0.11
Enwiki	4	1	14.60	34.90	1.91	9.55	6.24	3.04	5.10	2.89	-1.14
Enwiki	4	2	7.51	34.90	.90	5.16	26.40	.64	2.37	0.85	0.27
Enwiki	4	3	5.49	26.30	.60	3.44	18.40	.46	2.06	0.80	0.15
Enwiki	4	4	4.63	22.30	.43	2.76	17.90	.29	1.88	0.45	0.14
Enwiki	16	1	10.90	24.00	1.49	9.05	6.77	1.97	1.85	1.73	-0.47
Enwiki	16	2	5.79	16.20	.71	4.71	12.10	.60	1.09	0.41	0.10
Enwiki	16	3	4.12	11.60	.45	3.08	7.05	.45	1.05	0.46	0.00
Enwiki	16	4	3.71	8.76	.24	2.43	7.31	.25	1.29	0.15	-0.01
Enwiki	32	1	10.30	25.00	1.64	8.96	9.94	1.64	1.31	1.52	0.01
Enwiki	32	2	5.61	13.30	.72	4.61	10.00	.77	1.01	0.34	-0.05
Enwiki	32	3	4.22	9.22	.48	3.04	5.98	.61	1.19	0.33	-0.13
Enwiki	32	4	3.46	6.68	.30	2.50	5.07	.32	0.98	0.16	-0.02
Ruwiki	4	1	25.50	61.70	3.91	17.20	15.30	3.65	4.75	2.63	0.15
Ruwiki	4	2	13.30	69.20	1.68	9.45	53.20	1.15	2.19	0.91	0.30
Ruwiki	4	3	9.91	48.00	1.07	6.20	34.30	.86	2.10	0.78	0.11
Ruwiki	4	4	8.86	38.90	.74	4.87	32.80	.56	2.26	0.34	0.10
Ruwiki	16	1	19.60	44.40	2.61	16.20	17.20	3.02	1.94	1.54	-0.23
Ruwiki	16	2	10.70	30.30	1.26	8.54	23.50	1.22	1.20	0.39	0.03
Ruwiki	16	3	7.15	20.70	.80	5.56	15.00	.87	0.90	0.32	-0.04
Ruwiki	16	4	6.54	16.90	.46	4.36	13.80	.46	1.23	0.17	0.00
Ruwiki	32	1	18.60	44.10	3.01	16.10	19.50	3.22	1.41	1.39	-0.12
Ruwiki	32	2	9.82	26.00	1.47	8.36	20.60	1.19	0.83	0.30	0.16
Ruwiki	32	3	7.32	16.60	.93	5.43	11.50	1.06	1.07	0.29	-0.07
Ruwiki	32	4	6.16	12.30	.62	4.20	10.40	.61	1.11	0.11	0.01
Zhwiki	4	1	10.00	23.70	1.47	6.71	5.77	1.44	4.82	2.63	0.04
Zhwiki	4	2	5.14	27.10	.66	3.65	19.40	.47	2.19	1.13	0.28
Zhwiki	4	3	3.83	19.30	.42	2.44	13.60	.34	2.04	0.84	0.12
Zhwiki	4	4	3.32	15.50	.31	1.91	12.40	.21	2.08	0.46	0.14
Zhwiki	16	1				6.34	6.47	1.20			
Zhwiki	16	2	4.41	10.40	.46	3.33	8.79	.45	1.59	0.23	0.01
Zhwiki	16	3	2.94	8.07	.32	2.18	5.71	.34	1.12	0.35	-0.03
Zhwiki	16	4	2.73	6.49	.19	1.71	5.24	.20	1.51	0.18	-0.02
Zhwiki	32	1	7.24	17.20	1.24	6.29	8.01	1.25	1.40	1.35	-0.02
Zhwiki	32	2	4.09	9.36	.53	3.24	7.45	.54	1.24	0.28	-0.02
Zhwiki	32	3	2.84	6.54	.39	2.15	4.64	.41	1.02	0.28	-0.03
Zhwiki	32	4	2.52	4.86	.24	1.66	3.79	.27	1.27	0.16	-0.03
								μ	**1.78**	0.77	-0.00
								σ	1.04	0.73	0.22

Table D.105: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=Email, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	9.13	297.00	30.90	4.65	78.40	31.70	3.13	15.30	-0.59
Arwiki	4	2	4.84	345.00	16.00	3.10	209.00	16.00	1.21	9.48	0.04
Arwiki	4	3	3.36	272.00	10.50	2.23	164.00	10.60	0.79	7.54	-0.00
Arwiki	4	4	2.69	223.00	7.91	1.73	132.00	7.91	0.67	6.33	-0.00
Arwiki	16	1	5.82	234.00	31.90	4.38	110.00	32.10	1.01	8.68	-0.10
Arwiki	16	2	3.02	185.00	16.10	2.41	113.00	16.10	0.42	4.98	-0.05
Arwiki	16	3	2.07	131.00	10.60	1.60	80.60	10.50	0.33	3.54	0.04
Arwiki	16	4	1.69	107.00	7.94	1.32	68.80	7.98	0.26	2.65	-0.03
Arwiki	32	1	5.16	283.00	31.60	4.37	139.00	31.90	0.55	10.00	-0.26
Arwiki	32	2	2.69	173.00	15.80	2.37	103.00	16.00	0.23	4.87	-0.16
Arwiki	32	3	1.88	118.00	10.50	1.57	68.90	10.50	0.22	3.40	-0.01
Arwiki	32	4	1.50	91.20	7.86	1.25	55.30	7.93	0.18	2.50	-0.05
Enwiki	4	1	6.58	226.00	21.20	3.24	46.50	21.80	3.37	18.10	-0.69
Enwiki	4	2	3.40	233.00	11.10	2.11	145.00	11.00	1.30	8.92	0.10
Enwiki	4	3	2.34	185.00	7.27	1.52	115.00	7.24	0.82	7.01	0.03
Enwiki	4	4	1.87	152.00	5.44	1.19	92.80	5.43	0.69	6.02	0.00
Enwiki	16	1	4.02	156.00	21.90	3.04	53.30	22.10	0.99	10.40	-0.20
Enwiki	16	2	2.10	123.00	11.10	1.68	75.40	11.10	0.43	4.78	-0.01
Enwiki	16	3	1.47	86.40	7.30	1.11	48.20	7.25	0.36	3.86	0.05
Enwiki	16	4	1.17	69.80	5.46	.93	43.40	5.48	0.24	2.66	-0.02
Enwiki	32	1	3.54	193.00	21.70	3.06	81.50	22.00	0.48	11.20	-0.31
Enwiki	32	2	1.89	116.00	10.90	1.65	64.40	11.00	0.24	5.24	-0.14
Enwiki	32	3	1.32	80.30	7.22	1.10	41.90	7.27	0.22	3.86	-0.05
Enwiki	32	4	1.08	62.70	5.39	.88	34.70	5.46	0.20	2.82	-0.07
Ruwiki	4	1	11.30	371.00	38.30	5.82	91.70	38.80	3.11	15.80	-0.30
Ruwiki	4	2	6.03	430.00	19.80	3.88	257.00	19.70	1.22	9.80	0.09
Ruwiki	4	3	4.20	332.00	13.10	2.78	203.00	13.00	0.81	7.30	0.06
Ruwiki	4	4	3.26	275.00	9.81	2.17	164.00	9.80	0.62	6.32	0.01
Ruwiki	16	1	7.22	287.00	39.60	5.45	114.00	39.40	1.01	9.78	0.09
Ruwiki	16	2	3.78	226.00	20.10	3.07	140.00	19.90	0.40	4.89	0.10
Ruwiki	16	3	2.51	160.00	13.20	1.99	97.60	13.10	0.29	3.54	0.06
Ruwiki	16	4	2.00	129.00	9.84	1.66	84.90	9.92	0.19	2.49	-0.05
Ruwiki	32	1	6.45	335.00	39.30	5.43	150.00	39.30	0.58	10.50	0.03
Ruwiki	32	2	3.40	207.00	19.80	2.92	120.00	19.90	0.27	4.97	-0.03
Ruwiki	32	3	2.25	140.00	13.10	1.95	79.80	13.10	0.17	3.42	-0.00
Ruwiki	32	4	1.92	111.00	9.81	1.55	64.50	9.90	0.21	2.63	-0.05
Zhwiki	4	1	4.44	140.00	14.80	2.28	35.00	15.00	3.17	15.40	-0.29
Zhwiki	4	2	2.45	148.00	7.79	1.50	98.30	7.63	1.40	7.24	0.23
Zhwiki	4	3	1.74	121.00	5.17	1.09	78.40	5.05	0.96	6.24	0.17
Zhwiki	4	4	1.36	105.00	3.87	.86	64.00	3.84	0.74	6.01	0.05
Zhwiki	16	1	2.82	109.00	15.40	2.13	44.80	15.20	1.02	9.47	0.34
Zhwiki	16	2	1.52	88.50	7.86	1.19	54.80	7.71	0.48	4.94	0.21
Zhwiki	16	3	1.05	62.40	5.19	.79	37.90	5.11	0.37	3.60	0.12
Zhwiki	16	4	.84	49.70	3.90	.66	32.80	3.86	0.27	2.48	0.05
Zhwiki	32	1	2.53	130.00	15.30	2.13	58.30	15.20	0.59	10.60	0.19
Zhwiki	32	2	1.37	80.30	7.77	1.15	46.60	7.72	0.32	4.95	0.08
Zhwiki	32	3	.91	55.30	5.16	.79	31.30	5.11	0.17	3.52	0.07
Zhwiki	32	4	.80	43.80	3.89	.64	25.30	3.83	0.23	2.72	0.09
								μ	**0.77**	6.72	-0.02
								σ	0.81	3.86	0.18

Table D.106: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=Email, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	6.76	23.50	22.60	2.84	23.30	22.50	2.74	0.01	0.07
Arwiki	4	2	4.57	116.00	11.50	2.71	101.00	11.40	1.30	1.02	0.07
Arwiki	4	3	3.47	101.00	7.64	2.14	82.60	7.58	0.93	1.25	0.04
Arwiki	4	4	2.86	87.50	5.73	1.76	68.00	5.68	0.76	1.36	0.03
Arwiki	16	1	3.82	23.40	22.70	2.57	23.30	22.50	0.88	0.01	0.08
Arwiki	16	2	2.28	64.70	11.50	1.75	54.60	11.40	0.38	0.70	0.12
Arwiki	16	3	1.57	45.30	7.72	1.18	36.40	7.61	0.27	0.62	0.08
Arwiki	16	4	1.34	41.10	5.77	1.08	33.90	5.72	0.18	0.50	0.04
Arwiki	32	1	3.24	23.50	22.70	2.54	23.30	22.60	0.48	0.01	0.07
Arwiki	32	2	1.89	45.50	11.60	1.60	38.70	11.40	0.20	0.47	0.14
Arwiki	32	3	1.25	31.60	7.77	1.10	27.40	7.64	0.11	0.29	0.09
Arwiki	32	4	1.02	25.40	5.84	.93	22.50	5.76	0.07	0.20	0.06
Enwiki	4	1	4.91	15.90	15.60	2.03	15.90	15.60	2.90	0.00	0.01
Enwiki	4	2	3.22	80.50	7.90	1.90	72.30	7.81	1.33	0.83	0.09
Enwiki	4	3	2.40	69.90	5.25	1.50	58.80	5.22	0.91	1.12	0.03
Enwiki	4	4	1.98	60.90	3.93	1.23	48.30	3.90	0.76	1.26	0.03
Enwiki	16	1	2.67	15.90	15.60	1.79	15.80	15.60	0.89	0.01	0.02
Enwiki	16	2	1.56	40.60	7.92	1.19	36.80	7.83	0.37	0.38	0.09
Enwiki	16	3	1.06	29.20	5.29	.82	23.80	5.24	0.24	0.55	0.05
Enwiki	16	4	.92	25.60	3.97	.75	22.00	3.93	0.17	0.36	0.04
Enwiki	32	1	2.23	16.00	15.60	1.78	15.90	15.60	0.45	0.01	0.02
Enwiki	32	2	1.28	26.70	7.99	1.13	23.10	7.85	0.15	0.36	0.14
Enwiki	32	3	.86	18.10	5.33	.78	16.10	5.25	0.08	0.19	0.08
Enwiki	32	4	.71	14.60	4.01	.68	13.50	3.96	0.03	0.11	0.05
Ruwiki	4	1	8.40	29.40	28.00	3.57	29.10	27.70	2.73	0.02	0.12
Ruwiki	4	2	5.68	143.00	14.20	3.40	125.00	14.00	1.29	0.99	0.11
Ruwiki	4	3	4.32	123.00	9.47	2.67	102.00	9.38	0.94	1.21	0.05
Ruwiki	4	4	3.53	108.00	7.11	2.18	83.30	7.04	0.76	1.43	0.04
Ruwiki	16	1	4.76	29.40	28.00	3.23	29.00	27.80	0.87	0.02	0.13
Ruwiki	16	2	2.85	77.60	14.30	2.18	64.40	14.10	0.38	0.74	0.14
Ruwiki	16	3	1.93	57.50	9.58	1.48	47.30	9.42	0.26	0.57	0.09
Ruwiki	16	4	1.68	49.70	7.19	1.33	41.20	7.07	0.20	0.48	0.07
Ruwiki	32	1	4.06	29.50	28.10	3.18	29.00	27.90	0.49	0.03	0.11
Ruwiki	32	2	2.36	51.70	14.40	1.99	46.70	14.20	0.21	0.28	0.16
Ruwiki	32	3	1.55	35.40	9.65	1.35	31.50	9.48	0.11	0.22	0.10
Ruwiki	32	4	1.26	28.60	7.26	1.14	26.50	7.15	0.07	0.12	0.06
Zhwiki	4	1	3.27	11.70	10.90	1.39	11.50	10.80	2.76	0.03	0.25
Zhwiki	4	2	2.22	56.00	5.56	1.32	48.50	5.46	1.31	1.10	0.15
Zhwiki	4	3	1.68	48.80	3.72	1.05	39.50	3.66	0.93	1.37	0.09
Zhwiki	4	4	1.39	42.10	2.79	.87	32.50	2.76	0.76	1.41	0.05
Zhwiki	16	1	1.85	11.70	10.90	1.26	11.50	10.80	0.86	0.04	0.20
Zhwiki	16	2	1.12	30.50	5.62	.86	25.80	5.46	0.38	0.68	0.23
Zhwiki	16	3	.77	22.30	3.76	.61	18.40	3.67	0.24	0.58	0.13
Zhwiki	16	4	.67	19.30	2.83	.56	16.10	2.75	0.17	0.46	0.12
Zhwiki	32	1	1.58	11.80	11.00	1.26	11.50	10.90	0.48	0.04	0.18
Zhwiki	32	2	.94	20.60	5.66	.81	18.40	5.52	0.18	0.32	0.21
Zhwiki	32	3	.64	14.30	3.80	.58	12.50	3.70	0.08	0.27	0.16
Zhwiki	32	4	.54	11.60	2.87	.52	10.50	2.79	0.02	0.17	0.12
								$\boldsymbol{\mu}$	**0.69**	0.51	0.10
								$\boldsymbol{\sigma}$	0.73	0.46	0.06

Table D.107: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=Email, Colours=always

Arguments: URIOrEmail -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	36.70	199.00	5.54	28.50	114.00	8.96	5.70	5.88	-2.39
Arwiki	4	2	20.20	136.00	3.26	16.40	96.20	4.96	2.64	2.79	-1.19
Arwiki	4	3	15.40	98.10	2.82	12.10	71.60	4.47	2.34	1.85	-1.15
Arwiki	4	4	13.70	87.20	4.49	9.89	57.90	3.11	2.64	2.05	0.96
Arwiki	16	1	29.40	120.00	10.80	26.20	86.20	10.10	2.24	2.35	0.48
Arwiki	16	2	17.40	66.00	6.38	14.20	52.10	5.95	2.28	0.97	0.30
Arwiki	16	3	13.60	49.10	5.52	10.00	38.40	4.71	2.50	0.75	0.57
Arwiki	16	4	10.40	38.90	4.28	7.97	31.50	4.28	1.69	0.52	0.00
Arwiki	32	1	27.70	90.50	12.50	25.60	72.00	10.70	1.47	1.29	1.28
Arwiki	32	2	15.80	52.20	8.40	13.50	42.70	7.35	1.65	0.66	0.74
Arwiki	32	3	13.00	36.10	5.74	9.34	33.30	6.89	2.57	0.20	-0.80
Arwiki	32	4	10.60	29.40	4.98	7.49	25.40	5.41	2.14	0.28	-0.30
Enwiki	4	1	15.50	41.70	1.76	10.40	6.49	2.84	5.17	3.54	-1.08
Enwiki	4	2	8.16	35.80	.75	5.53	26.40	.63	2.65	0.95	0.12
Enwiki	4	3	5.71	27.20	.52	3.68	17.80	.45	2.04	0.95	0.06
Enwiki	4	4	4.93	23.90	.37	2.88	17.30	.30	2.06	0.66	0.07
Enwiki	16	1	11.80	27.60	1.44	9.87	7.54	1.69	1.94	2.02	-0.25
Enwiki	16	2	6.58	17.10	.69	5.15	13.10	.64	1.44	0.40	0.05
Enwiki	16	3	4.60	12.40	.43	3.37	7.31	.45	1.24	0.52	-0.01
Enwiki	16	4	4.12	9.45	.29	2.64	7.56	.24	1.50	0.19	0.04
Enwiki	32	1	11.10	26.50	1.62	9.77	11.10	1.75	1.35	1.55	-0.14
Enwiki	32	2	5.99	14.80	.70	5.01	9.93	.70	0.99	0.49	-0.00
Enwiki	32	3	4.56	9.85	.46	3.32	6.40	.65	1.25	0.35	-0.19
Enwiki	32	4	3.73	7.37	.30	2.52	4.75	.32	1.22	0.26	-0.01
Ruwiki	4	1	44.30	208.00	3.45	35.10	135.00	10.30	5.23	4.11	-3.91
Ruwiki	4	2	23.80	151.00	1.57	20.20	114.00	5.33	2.07	2.11	-2.13
Ruwiki	4	3	21.90	115.00	4.39	14.80	84.60	5.08	4.02	1.73	-0.39
Ruwiki	4	4	16.40	85.00	1.57	12.30	70.00	3.90	2.32	0.85	-1.32
Ruwiki	16	1	36.40	144.00	12.50	32.60	105.00	11.60	2.15	2.21	0.52
Ruwiki	16	2	21.60	80.30	7.66	17.60	67.20	8.15	2.26	0.74	-0.28
Ruwiki	16	3	16.10	59.70	6.35	12.70	46.40	5.27	1.93	0.75	0.61
Ruwiki	16	4	12.90	47.00	5.14	10.30	37.20	4.42	1.49	0.56	0.41
Ruwiki	32	1	34.30	109.00	14.50	31.80	85.10	12.10	1.45	1.35	1.37
Ruwiki	32	2	18.60	64.30	9.74	16.90	50.40	8.19	0.99	0.79	0.88
Ruwiki	32	3	15.80	42.40	6.21	12.10	38.20	6.92	2.11	0.24	-0.40
Ruwiki	32	4	12.20	34.90	5.42	9.79	28.30	5.03	1.36	0.38	0.22
Zhwiki	4	1	18.20	85.90	1.64	14.50	55.40	4.54	5.43	4.49	-4.27
Zhwiki	4	2	9.60	62.90	.83	8.15	47.20	3.42	2.14	2.31	-3.80
Zhwiki	4	3	8.03	46.30	1.50	5.98	36.30	3.03	3.01	1.47	-2.24
Zhwiki	4	4	6.52	34.80	.65	4.97	28.70	2.11	2.28	0.89	-2.14
Zhwiki	16	1				13.40	42.00	5.07			
Zhwiki	16	2	8.81	32.80	3.45	7.36	25.50	2.81	2.14	1.07	0.94
Zhwiki	16	3	6.89	24.00	2.85	5.30	18.60	2.19	2.34	0.80	0.97
Zhwiki	16	4	5.89	19.00	2.41	4.24	15.60	2.20	2.41	0.49	0.30
Zhwiki	32	1	14.10	44.10	6.14	13.10	34.00	5.29	1.47	1.48	1.25
Zhwiki	32	2	8.20	24.40	3.61	7.06	20.30	3.28	1.67	0.60	0.48
Zhwiki	32	3	6.50	17.30	2.68	5.02	15.80	3.05	2.17	0.22	-0.55
Zhwiki	32	4	5.42	14.50	2.58	4.08	12.10	2.36	1.97	0.36	0.32
								μ	**2.28**	1.31	-0.34
								σ	1.10	1.21	1.31

Table D.108: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=URIOrEmail, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	14.40	1010.00	31.80	8.94	463.00	56.50	3.82	38.40	-17.20
Arwiki	4	2	8.23	756.00	21.50	6.12	418.00	35.00	1.47	23.60	-9.37
Arwiki	4	3	5.94	561.00	20.20	5.02	304.00	21.60	0.64	17.90	-0.99
Arwiki	4	4	4.84	450.00	20.50	4.28	247.00	19.40	0.39	14.10	0.78
Arwiki	16	1	9.95	811.00	50.70	8.50	506.00	56.50	1.01	21.20	-4.05
Arwiki	16	2	5.61	466.00	50.50	5.11	300.00	29.20	0.34	11.60	14.80
Arwiki	16	3	4.47	337.00	42.80	3.90	209.00	20.40	0.40	8.96	15.60
Arwiki	16	4	3.22	253.00	32.10	3.25	168.00	18.60	-0.03	5.93	9.41
Arwiki	32	1	9.62	748.00	104.00	8.37	488.00	56.10	0.87	18.10	33.70
Arwiki	32	2	5.38	401.00	65.40	4.71	269.00	35.80	0.46	9.22	20.70
Arwiki	32	3	3.98	274.00	49.80	3.50	187.00	29.70	0.33	6.13	14.00
Arwiki	32	4	3.32	212.00	41.90	2.98	142.00	26.60	0.23	4.88	10.70
Enwiki	4	1	6.96	242.00	21.10	3.57	50.20	21.90	3.42	19.40	-0.81
Enwiki	4	2	3.54	253.00	11.00	2.23	141.00	11.00	1.32	11.20	0.02
Enwiki	4	3	2.45	192.00	7.29	1.60	112.00	7.28	0.86	8.14	0.01
Enwiki	4	4	1.92	158.00	5.43	1.25	91.80	5.44	0.67	6.72	-0.02
Enwiki	16	1	4.39	188.00	22.00	3.36	60.20	22.10	1.03	12.90	-0.12
Enwiki	16	2	2.28	135.00	11.10	1.85	79.30	11.10	0.44	5.62	0.02
Enwiki	16	3	1.52	94.80	7.28	1.21	50.00	7.25	0.31	4.51	0.03
Enwiki	16	4	1.24	76.20	5.45	.99	44.50	5.47	0.26	3.19	-0.02
Enwiki	32	1	3.89	217.00	21.80	3.37	95.40	22.00	0.53	12.30	-0.20
Enwiki	32	2	2.01	128.00	10.90	1.78	68.10	11.00	0.23	6.01	-0.14
Enwiki	32	3	1.38	86.80	7.21	1.20	44.30	7.23	0.18	4.28	-0.02
Enwiki	32	4	1.13	67.90	5.39	.95	33.90	5.45	0.18	3.42	-0.07
Ruwiki	4	1	17.60	1210.00	39.20	11.10	562.00	67.70	3.73	37.00	-16.20
Ruwiki	4	2	9.87	905.00	23.70	7.72	503.00	33.60	1.22	22.80	-5.62
Ruwiki	4	3	7.29	676.00	25.70	5.92	379.00	30.80	0.78	16.80	-2.88
Ruwiki	4	4	6.35	555.00	29.90	5.31	302.00	20.80	0.59	14.30	5.14
Ruwiki	16	1	12.70	1030.00	77.00	10.50	600.00	67.70	1.26	24.20	5.26
Ruwiki	16	2	6.90	571.00	63.50	6.18	369.00	39.40	0.41	11.40	13.60
Ruwiki	16	3	5.27	412.00	54.60	4.72	257.00	27.20	0.31	8.78	15.50
Ruwiki	16	4	4.45	322.00	45.50	4.00	202.00	22.60	0.26	6.80	13.00
Ruwiki	32	1	11.80	904.00	123.00	10.30	573.00	66.80	0.84	18.80	31.60
Ruwiki	32	2	6.48	489.00	81.10	5.86	315.00	40.60	0.35	9.83	22.90
Ruwiki	32	3	4.91	334.00	60.30	4.41	220.00	32.90	0.29	6.50	15.50
Ruwiki	32	4	4.07	258.00	49.90	3.80	163.00	26.70	0.15	5.40	13.10
Zhwiki	4	1	7.21	512.00	16.10	4.59	241.00	28.20	3.85	39.70	-17.80
Zhwiki	4	2	4.06	368.00	10.30	3.19	205.00	14.50	1.27	23.80	-6.19
Zhwiki	4	3	3.01	276.00	11.20	2.55	152.00	10.40	0.68	18.20	1.22
Zhwiki	4	4	2.39	218.00	10.40	2.19	124.00	9.45	0.29	13.80	1.36
Zhwiki	16	1				4.35	248.00	28.20			
Zhwiki	16	2	2.99	238.00	30.50	2.62	151.00	15.70	0.55	12.80	21.70
Zhwiki	16	3	2.27	167.00	23.20	2.00	105.00	11.80	0.40	9.11	16.80
Zhwiki	16	4	1.87	129.00	19.50	1.72	83.60	9.80	0.23	6.71	14.20
Zhwiki	32	1	4.88	374.00	53.40	4.29	237.00	28.00	0.87	20.10	37.30
Zhwiki	32	2	2.81	195.00	32.70	2.56	127.00	14.70	0.36	10.00	26.30
Zhwiki	32	3	2.12	136.00	25.80	1.85	92.50	15.50	0.39	6.39	15.20
Zhwiki	32	4	1.82	106.00	22.50	1.66	68.60	11.90	0.24	5.54	15.60
								μ	**0.82**	13.30	6.88
								σ	0.95	8.87	12.50

Table D.109: `icgrep` PAPI comparison between 2020 and `CURR` on `AVX2` with arguments Expression=URIOrEmail, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	9.84	82.60	23.30	5.88	57.50	41.50	2.76	1.75	-12.70
Arwiki	4	2	7.56	208.00	18.00	5.86	155.00	25.80	1.19	3.72	-5.43
Arwiki	4	3	6.18	178.00	16.40	5.19	124.00	19.40	0.69	3.78	-2.08
Arwiki	4	4	5.51	154.00	14.60	4.99	100.00	13.60	0.37	3.74	0.68
Arwiki	16	1	6.69	108.00	32.80	5.57	83.80	41.80	0.78	1.66	-6.31
Arwiki	16	2	4.32	141.00	34.40	3.94	109.00	30.20	0.26	2.20	2.87
Arwiki	16	3	3.25	99.80	24.70	3.69	72.30	17.20	-0.31	1.92	5.24
Arwiki	16	4	2.79	83.80	20.70	3.30	63.00	15.00	-0.36	1.45	3.98
Arwiki	32	1	5.80	129.00	44.60	5.51	94.30	42.00	0.20	2.42	1.84
Arwiki	32	2	3.68	116.00	39.10	3.76	93.50	29.60	-0.05	1.59	6.62
Arwiki	32	3	3.66	88.50	35.80	2.96	69.90	24.90	0.49	1.30	7.61
Arwiki	32	4	3.41	68.60	28.20	2.78	55.40	20.50	0.44	0.93	5.37
Enwiki	4	1	5.05	16.00	15.60	2.17	15.90	15.60	2.90	0.00	0.01
Enwiki	4	2	3.27	84.10	7.89	1.95	72.10	7.82	1.33	1.21	0.07
Enwiki	4	3	2.45	72.60	5.25	1.54	58.90	5.22	0.92	1.38	0.02
Enwiki	4	4	2.03	62.70	3.94	1.27	48.70	3.91	0.77	1.42	0.03
Enwiki	16	1	2.82	16.00	15.60	1.93	15.90	15.60	0.90	0.01	0.03
Enwiki	16	2	1.66	42.80	7.93	1.28	37.40	7.84	0.38	0.54	0.09
Enwiki	16	3	1.11	30.80	5.29	.87	24.20	5.23	0.25	0.66	0.06
Enwiki	16	4	.95	26.80	3.97	.79	22.00	3.94	0.17	0.48	0.03
Enwiki	32	1	2.37	16.10	15.60	1.93	15.90	15.60	0.44	0.01	0.02
Enwiki	32	2	1.35	27.10	8.00	1.19	23.00	7.85	0.15	0.41	0.15
Enwiki	32	3	.91	18.40	5.34	.82	16.30	5.25	0.09	0.22	0.09
Enwiki	32	4	.74	14.80	4.01	.71	13.60	3.96	0.04	0.12	0.05
Ruwiki	4	1	12.20	101.00	29.00	7.28	66.80	49.50	2.80	1.94	-11.60
Ruwiki	4	2	9.44	258.00	26.10	6.96	189.00	31.00	1.40	3.89	-2.77
Ruwiki	4	3	7.43	216.00	20.70	6.34	149.00	20.00	0.61	3.80	0.38
Ruwiki	4	4	6.74	190.00	19.90	5.50	128.00	21.00	0.70	3.50	-0.64
Ruwiki	16	1	8.42	135.00	41.00	6.86	98.80	49.80	0.88	2.06	-4.96
Ruwiki	16	2	6.35	171.00	47.60	4.80	124.00	35.30	0.88	2.68	6.94
Ruwiki	16	3	5.13	133.00	39.50	4.37	94.80	22.20	0.43	2.17	9.78
Ruwiki	16	4	4.47	108.00	32.60	3.90	76.20	18.80	0.33	1.83	7.83
Ruwiki	32	1	7.21	162.00	55.00	6.79	112.00	49.90	0.24	2.85	2.85
Ruwiki	32	2	4.93	138.00	48.90	4.78	104.00	32.30	0.08	1.91	9.41
Ruwiki	32	3	3.69	96.80	35.90	3.92	77.80	25.40	-0.13	1.07	5.92
Ruwiki	32	4	3.33	76.40	29.30	3.61	61.70	20.90	-0.16	0.83	4.74
Zhwiki	4	1	4.93	43.30	11.20	3.04	28.50	20.70	2.76	2.17	-13.90
Zhwiki	4	2	3.70	104.00	9.12	2.91	76.90	13.10	1.16	3.99	-5.88
Zhwiki	4	3	3.00	89.60	8.86	2.60	61.40	9.35	0.59	4.14	-0.72
Zhwiki	4	4	2.86	76.70	8.82	2.29	52.50	9.36	0.85	3.55	-0.79
Zhwiki	16	1	3.45	58.20	17.50	2.88	42.20	20.80	0.85	2.35	-4.85
Zhwiki	16	2	2.34	69.90	18.50	2.07	53.00	14.70	0.40	2.48	5.62
Zhwiki	16	3	1.80	52.40	14.80	1.65	42.10	12.60	0.22	1.51	3.11
Zhwiki	16	4	1.60	42.50	12.00	1.59	33.20	9.73	0.02	1.37	3.33
Zhwiki	32	1	3.40	79.30	31.30	2.87	49.10	21.00	0.79	4.44	15.10
Zhwiki	32	2	2.20	64.00	26.40	2.01	45.10	14.40	0.28	2.77	17.60
Zhwiki	32	3	1.89	44.80	19.10	1.60	34.80	12.80	0.43	1.46	9.34
Zhwiki	32	4	1.79	34.70	15.10	1.48	28.40	11.20	0.47	0.94	5.70
								μ	**0.66**	1.93	1.46
								σ	0.76	1.23	6.14

Table D.110: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=URIOrEmail, Colours=always

Arguments: Xquote -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	40.70	196.00	2.67	32.10	117.00	9.02	6.03	5.50	-4.43
Arwiki	4	2	22.20	132.00	1.58	18.10	101.00	6.29	2.88	2.16	-3.29
Arwiki	4	3	16.60	97.90	1.46	13.50	70.80	3.56	2.14	1.89	-1.47
Arwiki	4	4	14.60	76.50	1.10	10.70	63.30	5.42	2.73	0.92	-3.02
Arwiki	16	1	33.30	128.00	11.40	29.80	90.90	10.20	2.47	2.62	0.83
Arwiki	16	2	19.00	71.60	6.66	15.90	55.10	6.52	2.13	1.15	0.10
Arwiki	16	3	14.30	52.70	5.84	11.10	42.30	6.26	2.21	0.72	-0.29
Arwiki	16	4	11.40	43.00	5.11	8.83	34.30	5.21	1.83	0.60	-0.07
Arwiki	32	1	30.70	79.10	5.64	29.10	75.70	10.70	1.10	0.24	-3.55
Arwiki	32	2	16.10	44.10	3.13	15.20	47.00	8.02	0.58	-0.21	-3.41
Arwiki	32	3	14.00	38.00	5.67	10.60	33.60	6.65	2.36	0.31	-0.68
Arwiki	32	4	11.10	30.10	4.77	8.29	28.20	6.43	1.92	0.13	-1.16
Enwiki	4	1	18.90	52.20	1.74	13.00	9.33	2.25	5.91	4.32	-0.50
Enwiki	4	2	9.51	44.80	.77	6.88	29.00	.69	2.65	1.59	0.08
Enwiki	4	3	6.81	33.60	.51	4.62	20.70	.50	2.21	1.31	0.01
Enwiki	4	4	6.42	26.90	.39	3.56	17.90	.38	2.88	0.91	0.01
Enwiki	16	1	14.70	36.50	1.66	12.30	12.80	1.64	2.35	2.38	0.02
Enwiki	16	2	7.88	21.50	.80	6.35	13.90	.75	1.53	0.77	0.05
Enwiki	16	3	5.86	15.40	.51	4.21	8.90	.53	1.66	0.66	-0.02
Enwiki	16	4	4.86	12.10	.33	3.25	7.35	.38	1.62	0.48	-0.05
Enwiki	32	1	13.90	32.60	1.92	12.30	16.40	1.83	1.58	1.64	0.09
Enwiki	32	2	7.82	17.20	.95	6.27	13.30	.91	1.56	0.39	0.04
Enwiki	32	3	5.93	12.10	.58	4.16	7.91	.65	1.79	0.43	-0.08
Enwiki	32	4	4.79	9.15	.43	3.29	6.40	.47	1.51	0.28	-0.04
Ruwiki	4	1	47.50	214.00	3.54	37.50	128.00	9.54	5.70	4.87	-3.40
Ruwiki	4	2	25.70	150.00	1.81	21.10	108.00	4.87	2.58	2.42	-1.73
Ruwiki	4	3	20.00	107.00	1.43	15.50	78.70	4.06	2.56	1.62	-1.49
Ruwiki	4	4	17.40	87.40	1.67	12.70	65.30	3.44	2.64	1.25	-1.00
Ruwiki	16	1	38.40	124.00	5.43	34.90	103.00	10.70	1.99	1.17	-2.99
Ruwiki	16	2	22.30	82.60	7.31	18.90	60.90	5.83	1.96	1.23	0.84
Ruwiki	16	3	16.80	60.30	6.03	13.30	45.00	5.50	1.97	0.86	0.30
Ruwiki	16	4	15.10	46.90	4.74	10.90	36.00	4.51	2.36	0.61	0.13
Ruwiki	32	1	37.00	111.00	13.90	34.10	84.40	11.50	1.64	1.50	1.36
Ruwiki	32	2	20.50	62.50	8.55	17.90	52.30	8.13	1.47	0.58	0.24
Ruwiki	32	3	16.20	43.40	6.00	12.70	36.10	6.39	1.98	0.41	-0.22
Ruwiki	32	4	13.70	34.50	4.98	9.95	31.30	6.68	2.14	0.18	-0.96
Zhwiki	4	1	19.30	86.80	1.26	15.40	51.50	4.27	5.73	5.18	-4.42
Zhwiki	4	2	10.90	57.60	.71	8.66	43.80	2.74	3.25	2.02	-2.98
Zhwiki	4	3	8.31	46.70	1.26	6.50	32.10	1.62	2.65	2.15	-0.53
Zhwiki	4	4	7.41	39.40	1.98	5.20	27.60	2.21	3.25	1.73	-0.34
Zhwiki	16	1				14.30	41.70	4.77			
Zhwiki	16	2	8.80	34.00	3.23	7.75	25.20	2.75	1.53	1.30	0.70
Zhwiki	16	3	7.11	23.60	2.35	5.58	17.40	1.88	2.25	0.92	0.69
Zhwiki	16	4	6.02	19.40	2.22	4.42	15.10	2.07	2.35	0.64	0.23
Zhwiki	32	1	15.10	44.90	6.08	13.90	33.70	5.02	1.68	1.64	1.56
Zhwiki	32	2	8.15	25.90	4.01	7.54	20.00	2.77	0.90	0.87	1.81
Zhwiki	32	3	6.75	17.60	2.60	5.24	15.40	3.02	2.22	0.33	-0.60
Zhwiki	32	4	5.70	14.30	2.36	4.43	11.10	1.89	1.86	0.48	0.69
								μ	**2.39**	1.39	-0.70
								σ	1.19	1.29	1.55

Table D.111: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=Xquote, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	16.10	1080.00	32.30	10.20	496.00	57.70	4.11	40.60	-17.70
Arwiki	4	2	8.90	801.00	21.60	6.98	427.00	30.00	1.34	26.10	-5.88
Arwiki	4	3	6.43	596.00	21.10	5.55	311.00	20.80	0.61	19.90	0.19
Arwiki	4	4	5.24	478.00	21.30	4.73	251.00	18.60	0.36	15.80	1.93
Arwiki	16	1	11.80	924.00	66.70	9.72	542.00	58.10	1.44	26.70	6.02
Arwiki	16	2	6.74	520.00	59.90	5.67	327.00	33.10	0.74	13.50	18.60
Arwiki	16	3	4.75	373.00	50.50	4.15	236.00	28.40	0.42	9.53	15.40
Arwiki	16	4	4.04	283.00	38.40	3.50	186.00	23.80	0.38	6.78	10.20
Arwiki	32	1	11.00	816.00	105.00	9.59	550.00	57.00	1.00	18.50	33.80
Arwiki	32	2	6.14	427.00	63.00	5.32	306.00	37.70	0.57	8.46	17.60
Arwiki	32	3	4.50	296.00	50.20	3.96	211.00	30.90	0.38	5.95	13.50
Arwiki	32	4	3.65	230.00	43.50	3.24	159.00	30.80	0.28	4.95	8.86
Enwiki	4	1	8.36	264.00	22.10	4.46	68.00	22.10	3.93	19.80	0.03
Enwiki	4	2	4.23	297.00	11.60	2.72	158.00	11.30	1.52	14.00	0.32
Enwiki	4	3	2.93	222.00	7.67	1.92	121.00	7.45	1.01	10.20	0.21
Enwiki	4	4	2.30	185.00	5.80	1.53	99.60	5.67	0.77	8.58	0.13
Enwiki	16	1	5.53	255.00	23.10	4.22	92.20	22.40	1.33	16.40	0.73
Enwiki	16	2	2.79	169.00	11.80	2.27	92.90	11.40	0.53	7.70	0.42
Enwiki	16	3	2.08	122.00	7.81	1.54	65.30	7.48	0.54	5.67	0.33
Enwiki	16	4	1.66	94.50	5.89	1.22	53.20	5.75	0.44	4.17	0.14
Enwiki	32	1	5.02	285.00	23.20	4.24	148.00	22.20	0.79	13.80	0.97
Enwiki	32	2	2.57	158.00	11.80	2.22	93.50	11.30	0.35	6.47	0.48
Enwiki	32	3	1.87	112.00	7.75	1.51	61.10	7.45	0.36	5.16	0.31
Enwiki	32	4	1.53	87.30	5.88	1.19	44.30	5.74	0.34	4.33	0.14
Ruwiki	4	1	19.10	1210.00	39.80	12.10	556.00	65.60	4.00	37.10	-14.60
Ruwiki	4	2	10.50	910.00	24.40	8.05	491.00	34.10	1.38	23.70	-5.53
Ruwiki	4	3	7.59	683.00	24.60	6.19	368.00	28.00	0.79	17.80	-1.95
Ruwiki	4	4	6.15	550.00	22.20	5.43	288.00	19.60	0.41	14.80	1.49
Ruwiki	16	1	13.50	1010.00	61.00	11.40	598.00	65.90	1.16	23.20	-2.77
Ruwiki	16	2	7.85	589.00	60.80	6.46	375.00	41.80	0.79	12.10	10.80
Ruwiki	16	3	4.86	396.00	39.00	5.03	257.00	25.30	-0.10	7.89	7.74
Ruwiki	16	4	4.73	321.00	40.10	4.00	212.00	28.00	0.42	6.21	6.83
Ruwiki	32	1	12.70	913.00	104.00	11.30	612.00	64.50	0.80	17.00	22.40
Ruwiki	32	2	7.08	497.00	71.20	6.42	333.00	35.10	0.37	9.33	20.40
Ruwiki	32	3	5.15	347.00	57.50	4.85	223.00	26.40	0.17	7.05	17.70
Ruwiki	32	4	4.28	265.00	46.20	3.95	168.00	28.00	0.18	5.47	10.30
Zhwiki	4	1	7.69	486.00	15.60	5.00	229.00	27.60	3.96	37.80	-17.60
Zhwiki	4	2	4.30	367.00	10.70	3.40	200.00	13.90	1.32	24.60	-4.81
Zhwiki	4	3	3.15	274.00	10.80	2.67	147.00	10.50	0.70	18.60	0.44
Zhwiki	4	4	2.57	217.00	10.00	2.30	118.00	8.93	0.40	14.60	1.57
Zhwiki	16	1				4.74	245.00	27.80			
Zhwiki	16	2	3.29	241.00	27.40	2.82	151.00	14.70	0.69	13.10	18.60
Zhwiki	16	3	2.06	160.00	17.20	2.09	107.00	12.60	-0.05	7.79	6.70
Zhwiki	16	4	2.01	131.00	18.00	1.80	83.90	9.91	0.31	6.90	11.90
Zhwiki	32	1	5.15	370.00	43.40	4.68	252.00	27.30	0.68	17.20	23.70
Zhwiki	32	2	2.92	205.00	32.50	2.66	139.00	17.00	0.39	9.62	22.80
Zhwiki	32	3	2.00	135.00	21.30	2.08	92.00	11.40	-0.13	6.28	14.50
Zhwiki	32	4	1.84	108.00	20.20	1.78	67.00	10.50	0.08	6.00	14.20
								μ	**0.90**	14.00	5.78
								σ	1.03	8.95	10.70

Table D.112: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=Xquote, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	10.90	83.40	23.40	6.57	66.50	42.10	3.01	1.17	-13.10
Arwiki	4	2	8.00	217.00	18.70	6.25	165.00	26.80	1.22	3.63	-5.66
Arwiki	4	3	6.61	187.00	17.80	5.36	133.00	21.10	0.87	3.73	-2.34
Arwiki	4	4	5.50	159.00	13.90	5.23	107.00	13.90	0.19	3.63	0.02
Arwiki	16	1	7.27	108.00	29.70	6.22	87.30	42.70	0.73	1.42	-9.06
Arwiki	16	2	4.49	146.00	31.80	4.66	108.00	26.60	-0.12	2.59	3.63
Arwiki	16	3	3.35	104.00	24.30	3.96	76.20	18.20	-0.42	1.96	4.25
Arwiki	16	4	3.32	90.00	22.50	3.37	67.80	17.80	-0.04	1.55	3.30
Arwiki	32	1	7.45	158.00	61.90	6.11	98.30	42.80	0.93	4.15	13.40
Arwiki	32	2	5.10	128.00	46.40	4.27	95.30	28.20	0.58	2.27	12.70
Arwiki	32	3	4.06	91.80	34.90	3.40	70.90	23.70	0.46	1.46	7.77
Arwiki	32	4	3.65	72.60	28.90	3.03	58.00	21.00	0.43	1.02	5.51
Enwiki	4	1	5.89	17.80	16.00	2.61	17.50	15.80	3.31	0.03	0.22
Enwiki	4	2	3.81	99.10	8.26	2.27	80.10	8.06	1.55	1.92	0.20
Enwiki	4	3	2.85	84.10	5.52	1.76	65.70	5.41	1.09	1.86	0.11
Enwiki	4	4	2.37	73.60	4.14	1.45	54.50	4.08	0.93	1.92	0.06
Enwiki	16	1	3.46	18.00	16.20	2.36	17.40	15.80	1.10	0.06	0.37
Enwiki	16	2	2.02	50.80	8.39	1.50	38.70	8.11	0.52	1.22	0.28
Enwiki	16	3	1.37	36.60	5.64	1.04	26.50	5.46	0.34	1.02	0.18
Enwiki	16	4	1.17	32.50	4.24	.91	24.30	4.14	0.26	0.83	0.10
Enwiki	32	1	2.97	18.50	16.30	2.33	17.50	16.00	0.65	0.09	0.34
Enwiki	32	2	1.67	31.40	8.55	1.41	25.20	8.22	0.26	0.63	0.33
Enwiki	32	3	1.13	21.80	5.74	.99	18.20	5.53	0.14	0.36	0.22
Enwiki	32	4	.92	17.80	4.33	.83	15.50	4.17	0.09	0.23	0.16
Ruwiki	4	1	13.00	94.80	28.60	7.65	64.90	47.80	3.03	1.69	-10.90
Ruwiki	4	2	9.41	256.00	22.80	6.85	195.00	32.60	1.45	3.46	-5.50
Ruwiki	4	3	7.43	215.00	18.70	6.33	152.00	18.80	0.62	3.58	-0.08
Ruwiki	4	4	6.52	189.00	16.60	5.27	133.00	22.00	0.71	3.18	-3.08
Ruwiki	16	1	8.64	116.00	34.60	7.23	92.80	48.20	0.80	1.32	-7.68
Ruwiki	16	2	5.61	162.00	37.20	4.99	119.00	33.00	0.35	2.42	2.40
Ruwiki	16	3	4.17	123.00	29.10	4.31	93.10	22.30	-0.08	1.70	3.84
Ruwiki	16	4	3.81	102.00	24.50	3.47	79.70	23.40	0.20	1.28	0.60
Ruwiki	32	1	8.23	161.00	61.20	7.10	103.00	48.40	0.64	3.29	7.27
Ruwiki	32	2	5.57	141.00	54.00	4.94	99.60	30.00	0.36	2.32	13.60
Ruwiki	32	3	4.63	99.40	38.00	3.62	79.20	29.10	0.57	1.14	5.01
Ruwiki	32	4	4.11	78.20	30.60	3.63	60.30	19.90	0.28	1.02	6.05
Zhwiki	4	1	5.25	38.90	11.40	3.23	30.80	20.10	2.97	1.19	-12.70
Zhwiki	4	2	3.79	103.00	9.34	3.06	76.10	11.60	1.07	3.89	-3.33
Zhwiki	4	3	3.04	87.80	8.37	2.72	60.50	7.90	0.47	4.01	0.68
Zhwiki	4	4	2.76	75.50	7.38	2.41	51.40	7.66	0.53	3.54	-0.41
Zhwiki	16	1	3.50	49.50	14.00	3.05	39.80	20.30	0.66	1.42	-9.17
Zhwiki	16	2	2.21	66.30	15.30	2.22	50.00	13.30	-0.02	2.40	2.89
Zhwiki	16	3	1.89	51.20	13.30	1.97	37.30	8.69	-0.12	2.05	6.73
Zhwiki	16	4	1.49	40.90	9.90	1.76	30.30	7.64	-0.38	1.57	3.33
Zhwiki	32	1	3.25	65.90	23.20	3.01	44.70	20.40	0.34	3.10	4.04
Zhwiki	32	2	1.98	54.50	18.50	2.26	40.70	11.50	-0.41	2.02	10.30
Zhwiki	32	3	1.91	43.10	17.70	1.90	29.90	8.94	0.02	1.93	12.90
Zhwiki	32	4	1.85	32.90	13.30	1.68	25.20	8.34	0.24	1.14	7.27
								μ	**0.67**	1.95	1.19
								σ	0.85	1.13	6.29

Table D.113: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=Xquote, Colours=always

Arguments: \p{Greek} -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	20.40	48.70	3.05	13.10	13.20	3.10	5.14	2.48	-0.03
Arwiki	4	2	10.90	54.30	1.40	7.28	44.10	.94	2.53	0.71	0.32
Arwiki	4	3	8.05	39.20	.82	4.79	29.80	.74	2.28	0.66	0.06
Arwiki	4	4	6.73	33.60	.63	3.82	27.60	.45	2.03	0.42	0.13
Arwiki	16	1	15.50	37.20	2.28	12.40	13.80	2.62	2.17	1.63	-0.24
Arwiki	16	2	8.25	25.60	.98	6.47	17.90	.91	1.24	0.54	0.05
Arwiki	16	3	6.38	17.20	.61	4.30	12.20	.69	1.45	0.35	-0.06
Arwiki	16	4	5.31	13.60	.36	3.34	10.60	.39	1.37	0.21	-0.02
Arwiki	32	1	14.70	34.70	2.42	12.30	15.90	2.77	1.63	1.31	-0.24
Arwiki	32	2	8.08	19.60	1.21	6.36	13.90	1.11	1.20	0.40	0.07
Arwiki	32	3	5.46	13.50	.76	4.20	8.87	.87	0.88	0.32	-0.08
Arwiki	32	4	4.99	10.50	.60	3.21	6.96	.52	1.24	0.25	0.05
Enwiki	4	1	14.30	35.80	2.13	8.82	5.91	3.02	5.52	3.01	-0.90
Enwiki	4	2	7.53	34.40	.93	4.84	27.70	.65	2.71	0.67	0.29
Enwiki	4	3	5.16	28.40	.59	3.25	19.90	.48	1.92	0.85	0.12
Enwiki	4	4	4.29	24.00	.44	2.56	18.70	.30	1.73	0.53	0.14
Enwiki	16	1	10.50	24.70	1.59	8.37	6.34	2.14	2.16	1.85	-0.56
Enwiki	16	2	5.62	17.40	.66	4.35	11.90	.60	1.28	0.55	0.06
Enwiki	16	3	4.11	11.70	.43	2.86	7.33	.48	1.26	0.44	-0.04
Enwiki	16	4	3.58	9.26	.31	2.49	6.93	.27	1.10	0.23	0.04
Enwiki	32	1	9.89	24.00	1.75	8.28	8.60	1.75	1.62	1.55	-0.01
Enwiki	32	2	5.29	14.00	.70	4.27	9.02	.66	1.02	0.50	0.04
Enwiki	32	3	4.00	9.26	.51	2.81	5.46	.59	1.20	0.38	-0.09
Enwiki	32	4	3.35	6.59	.32	2.18	4.47	.35	1.18	0.21	-0.04
Ruwiki	4	1	25.80	65.80	3.96	16.80	18.80	4.00	5.07	2.66	-0.02
Ruwiki	4	2	13.70	73.20	2.03	9.42	56.40	1.31	2.42	0.95	0.40
Ruwiki	4	3	10.40	51.80	1.15	6.36	38.40	1.00	2.28	0.76	0.09
Ruwiki	4	4	8.44	43.60	.91	5.20	34.90	.68	1.84	0.49	0.13
Ruwiki	16	1	19.80	50.90	3.10	16.00	20.10	3.43	2.17	1.74	-0.19
Ruwiki	16	2	11.00	32.90	1.46	8.37	23.00	1.44	1.47	0.56	0.01
Ruwiki	16	3	7.97	23.20	1.01	5.63	17.50	1.10	1.33	0.32	-0.05
Ruwiki	16	4	6.77	18.30	.69	4.40	14.50	.66	1.34	0.22	0.02
Ruwiki	32	1	18.70	46.00	3.68	15.80	21.80	3.68	1.62	1.37	0.00
Ruwiki	32	2	10.20	26.10	1.81	8.18	19.30	1.63	1.18	0.39	0.10
Ruwiki	32	3	7.38	18.00	1.29	5.45	12.60	1.37	1.09	0.30	-0.04
Ruwiki	32	4	6.42	13.60	.81	4.21	10.10	.86	1.25	0.20	-0.03
Zhwiki	4	1	10.20	23.70	1.51	6.72	6.62	1.58	5.17	2.51	-0.10
Zhwiki	4	2	5.34	28.60	.74	3.70	20.60	.50	2.41	1.18	0.35
Zhwiki	4	3	3.95	20.50	.46	2.51	14.70	.36	2.10	0.85	0.15
Zhwiki	4	4	3.33	17.70	.34	1.95	13.00	.28	2.02	0.69	0.08
Zhwiki	16	1	7.85	19.40	1.31	6.41	7.02	1.37	2.11	1.81	-0.09
Zhwiki	16	2	4.26	12.80	.58	3.36	8.58	.49	1.31	0.62	0.13
Zhwiki	16	3	3.05	8.97	.39	2.23	6.31	.39	1.20	0.39	-0.00
Zhwiki	16	4	2.59	7.09	.27	1.74	5.20	.24	1.25	0.28	0.05
Zhwiki	32	1	7.42	17.60	1.46	6.31	8.23	1.38	1.62	1.38	0.11
Zhwiki	32	2	4.00	10.30	.74	3.28	7.16	.56	1.06	0.46	0.26
Zhwiki	32	3	3.04	6.80	.48	2.18	4.64	.51	1.27	0.32	-0.04
Zhwiki	32	4	2.51	5.20	.30	1.67	3.64	.33	1.23	0.23	-0.04
								μ	**1.91**	0.85	0.01
								σ	1.10	0.72	0.20

Table D.114: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=\p{Greek}, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	9.13	270.00	31.20	4.43	74.50	31.60	3.28	13.70	-0.23
Arwiki	4	2	4.85	345.00	16.20	3.01	204.00	16.00	1.29	9.88	0.11
Arwiki	4	3	3.37	265.00	10.80	2.17	161.00	10.60	0.84	7.29	0.10
Arwiki	4	4	2.66	222.00	8.03	1.69	130.00	7.99	0.68	6.42	0.03
Arwiki	16	1	5.77	229.00	32.40	4.13	98.50	31.90	1.15	9.09	0.36
Arwiki	16	2	3.01	184.00	16.30	2.31	110.00	16.20	0.48	5.14	0.02
Arwiki	16	3	2.07	129.00	10.90	1.54	78.90	10.70	0.37	3.51	0.14
Arwiki	16	4	1.74	105.00	8.13	1.27	66.40	8.09	0.33	2.72	0.02
Arwiki	32	1	5.11	273.00	32.10	4.11	121.00	32.10	0.70	10.60	-0.02
Arwiki	32	2	2.67	164.00	16.20	2.22	97.10	16.20	0.31	4.67	0.03
Arwiki	32	3	1.84	113.00	10.80	1.49	65.20	10.70	0.24	3.35	0.03
Arwiki	32	4	1.50	88.90	8.04	1.19	51.50	8.06	0.22	2.61	-0.01
Enwiki	4	1	6.55	189.00	21.50	3.01	41.00	22.00	3.56	14.90	-0.50
Enwiki	4	2	3.37	230.00	11.20	2.02	142.00	11.10	1.36	8.84	0.11
Enwiki	4	3	2.34	178.00	7.41	1.47	113.00	7.33	0.87	6.48	0.09
Enwiki	4	4	1.87	148.00	5.50	1.16	92.70	5.50	0.71	5.56	0.00
Enwiki	16	1				2.80	48.80	22.20			
Enwiki	16	2	2.14	120.00	11.30	1.60	74.80	11.20	0.55	4.56	0.07
Enwiki	16	3	1.43	85.30	7.40	1.04	47.00	7.33	0.39	3.86	0.07
Enwiki	16	4	1.16	70.00	5.52	.88	42.70	5.55	0.29	2.75	-0.03
Enwiki	32	1	3.49	181.00	22.10	2.80	67.00	22.20	0.70	11.50	-0.17
Enwiki	32	2	1.91	110.00	11.10	1.52	60.10	11.20	0.39	5.00	-0.09
Enwiki	32	3	1.36	78.80	7.33	1.03	39.90	7.36	0.33	3.91	-0.03
Enwiki	32	4	1.07	60.90	5.48	.82	32.30	5.53	0.25	2.87	-0.05
Ruwiki	4	1	11.60	360.00	39.30	5.83	104.00	39.90	3.26	14.50	-0.33
Ruwiki	4	2	6.26	443.00	21.10	3.92	257.00	20.30	1.33	10.50	0.44
Ruwiki	4	3	4.40	343.00	14.20	2.85	203.00	13.50	0.88	7.93	0.41
Ruwiki	4	4	3.47	280.00	10.80	2.26	165.00	10.10	0.68	6.56	0.36
Ruwiki	16	1	7.47	313.00	42.20	5.43	131.00	40.20	1.15	10.30	1.16
Ruwiki	16	2	3.92	234.00	21.80	3.06	144.00	20.40	0.48	5.09	0.76
Ruwiki	16	3	2.65	170.00	14.50	2.06	103.00	13.70	0.34	3.83	0.44
Ruwiki	16	4	2.13	134.00	11.00	1.69	85.80	10.40	0.25	2.73	0.34
Ruwiki	32	1	6.68	345.00	42.50	5.38	144.00	40.30	0.73	11.40	1.25
Ruwiki	32	2	3.50	211.00	21.70	2.93	122.00	20.60	0.32	5.06	0.64
Ruwiki	32	3	2.38	144.00	14.80	1.98	81.80	13.80	0.22	3.50	0.55
Ruwiki	32	4	1.87	111.00	11.00	1.57	65.00	10.70	0.17	2.59	0.16
Zhwiki	4	1	4.64	136.00	15.30	2.37	36.70	15.30	3.33	14.50	0.03
Zhwiki	4	2	2.49	168.00	8.17	1.55	96.40	7.83	1.38	10.50	0.50
Zhwiki	4	3	1.77	128.00	5.50	1.13	76.00	5.14	0.94	7.67	0.53
Zhwiki	4	4	1.43	105.00	4.15	.90	62.50	3.91	0.77	6.27	0.35
Zhwiki	16	1	3.01	118.00	16.20	2.20	45.60	15.50	1.18	10.70	1.03
Zhwiki	16	2	1.60	90.80	8.30	1.24	55.00	7.80	0.53	5.25	0.72
Zhwiki	16	3	1.13	64.30	5.56	.84	38.60	5.24	0.42	3.77	0.47
Zhwiki	16	4	.92	52.20	4.21	.69	31.60	3.97	0.34	3.03	0.36
Zhwiki	32	1	2.70	130.00	16.30	2.19	54.30	15.50	0.74	11.10	1.10
Zhwiki	32	2	1.42	80.20	8.35	1.19	45.40	7.87	0.34	5.11	0.70
Zhwiki	32	3	1.01	55.60	5.67	.82	31.40	5.42	0.27	3.55	0.37
Zhwiki	32	4	.85	44.50	4.22	.67	24.50	4.05	0.27	2.93	0.26
								μ	**0.84**	6.76	0.27
								σ	0.84	3.64	0.38

Table D.115: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=\p{Greek}, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	6.80	24.30	22.70	2.80	24.00	22.50	2.79	0.02	0.14
Arwiki	4	2	4.64	115.00	11.60	2.75	102.00	11.40	1.31	0.89	0.12
Arwiki	4	3	3.49	98.70	7.74	2.16	81.70	7.64	0.93	1.18	0.07
Arwiki	4	4	2.88	85.50	5.80	1.77	67.10	5.73	0.77	1.29	0.05
Arwiki	16	1	3.88	24.30	22.80	2.52	23.90	22.60	0.95	0.03	0.17
Arwiki	16	2	2.33	66.10	11.70	1.74	53.70	11.50	0.41	0.87	0.18
Arwiki	16	3	1.58	46.10	7.82	1.19	35.60	7.67	0.28	0.73	0.10
Arwiki	16	4	1.36	41.70	5.88	1.08	32.90	5.76	0.20	0.61	0.08
Arwiki	32	1	3.30	24.40	22.90	2.48	24.00	22.70	0.57	0.02	0.15
Arwiki	32	2	1.93	45.10	11.80	1.57	38.20	11.50	0.25	0.48	0.16
Arwiki	32	3	1.28	31.40	7.91	1.08	26.90	7.74	0.14	0.32	0.12
Arwiki	32	4	1.06	25.40	5.96	.91	21.90	5.85	0.10	0.24	0.07
Enwiki	4	1	5.01	16.30	15.70	1.98	16.20	15.60	3.05	0.01	0.06
Enwiki	4	2	3.23	76.60	7.95	1.94	72.80	7.85	1.31	0.39	0.11
Enwiki	4	3	2.42	66.70	5.29	1.50	57.80	5.25	0.92	0.90	0.04
Enwiki	4	4	1.98	58.00	3.97	1.23	47.20	3.94	0.76	1.08	0.03
Enwiki	16	1	2.75	16.30	15.70	1.73	16.10	15.60	1.03	0.02	0.06
Enwiki	16	2	1.61	41.40	8.00	1.18	36.40	7.88	0.43	0.50	0.13
Enwiki	16	3	1.09	30.00	5.34	.81	23.50	5.26	0.28	0.65	0.07
Enwiki	16	4	.94	25.80	4.01	.76	21.60	3.95	0.18	0.42	0.06
Enwiki	32	1	2.30	16.40	15.70	1.72	16.20	15.70	0.59	0.02	0.08
Enwiki	32	2	1.33	26.80	8.05	1.09	22.90	7.91	0.24	0.40	0.14
Enwiki	32	3	.89	18.20	5.38	.76	16.00	5.29	0.13	0.23	0.10
Enwiki	32	4	.72	14.50	4.05	.67	13.40	4.00	0.06	0.11	0.04
Ruwiki	4	1	8.66	34.20	28.50	3.75	32.80	28.30	2.78	0.08	0.12
Ruwiki	4	2	5.98	147.00	15.30	3.59	127.00	14.60	1.36	1.14	0.40
Ruwiki	4	3	4.53	126.00	10.30	2.80	101.00	9.94	0.97	1.39	0.23
Ruwiki	4	4	3.77	109.00	7.82	2.31	82.70	7.47	0.82	1.50	0.20
Ruwiki	16	1	5.09	34.90	29.50	3.39	32.80	28.40	0.97	0.12	0.64
Ruwiki	16	2	3.07	82.70	15.70	2.30	64.40	14.70	0.44	1.04	0.58
Ruwiki	16	3	2.13	61.60	10.50	1.61	48.40	10.10	0.29	0.75	0.23
Ruwiki	16	4	1.81	52.30	8.00	1.40	40.70	7.58	0.23	0.66	0.23
Ruwiki	32	1	4.36	35.30	29.90	3.33	33.00	28.50	0.58	0.13	0.81
Ruwiki	32	2	2.54	54.60	15.90	2.08	47.00	15.20	0.26	0.43	0.43
Ruwiki	32	3	1.73	38.10	10.70	1.46	32.90	10.10	0.15	0.30	0.33
Ruwiki	32	4	1.42	30.40	8.14	1.21	27.30	7.90	0.12	0.17	0.13
Zhwiki	4	1	3.46	12.90	11.10	1.50	12.50	10.90	2.87	0.06	0.31
Zhwiki	4	2	2.32	55.50	5.85	1.40	48.30	5.59	1.34	1.07	0.37
Zhwiki	4	3	1.76	48.60	3.95	1.09	39.00	3.87	0.98	1.41	0.12
Zhwiki	4	4	1.46	42.00	2.97	.91	31.80	2.85	0.80	1.49	0.18
Zhwiki	16	1	2.03	13.10	11.40	1.36	12.50	11.00	0.98	0.09	0.66
Zhwiki	16	2	1.22	32.20	5.99	.93	25.60	5.67	0.43	0.98	0.47
Zhwiki	16	3	.85	23.50	4.04	.67	18.50	3.82	0.26	0.74	0.32
Zhwiki	16	4	.73	20.00	3.05	.59	15.70	2.91	0.20	0.64	0.21
Zhwiki	32	1	1.74	13.20	11.50	1.35	12.60	11.10	0.58	0.09	0.64
Zhwiki	32	2	1.02	21.40	6.09	.86	18.10	5.73	0.24	0.48	0.52
Zhwiki	32	3	.71	15.00	4.12	.63	12.60	3.87	0.11	0.35	0.36
Zhwiki	32	4	.60	12.10	3.12	.56	10.50	2.95	0.06	0.24	0.24
								$\boldsymbol{\mu}$	**0.74**	0.56	0.23
								$\boldsymbol{\sigma}$	0.75	0.45	0.19

Table D.116: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=\p{Greek}, Colours=always

Arguments: \X -colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	114.00	635.00	32.00	98.60	410.00	31.60	10.70	15.70	0.22
Arwiki	4	2	63.60	336.00	18.80	52.40	242.00	20.70	7.85	6.53	-1.34
Arwiki	4	3	47.10	254.00	20.30	37.20	171.00	15.30	6.88	5.75	3.54
Arwiki	4	4	38.90	206.00	16.70	29.00	139.00	14.10	6.92	4.66	1.79
Arwiki	16	1	96.00	245.00	33.90	91.40	212.00	33.20	3.23	2.32	0.53
Arwiki	16	2	53.40	148.00	25.90	47.70	121.00	22.30	4.02	1.86	2.45
Arwiki	16	3	39.10	110.00	21.40	32.60	93.90	20.70	4.56	1.14	0.50
Arwiki	16	4	31.40	88.80	17.80	26.50	67.60	14.00	3.46	1.48	2.64
Arwiki	32	1	92.80	201.00	36.10	89.90	168.00	31.50	2.06	2.34	3.25
Arwiki	32	2	52.20	116.00	22.70	46.90	97.10	21.30	3.70	1.28	0.98
Arwiki	32	3	42.60	84.80	19.30	32.00	77.00	21.40	7.37	0.55	-1.45
Arwiki	32	4	30.40	74.20	19.00	25.30	59.30	17.10	3.58	1.04	1.32
Enwiki	4	1	81.80	446.00	30.30	70.40	293.00	25.80	11.50	15.40	4.48
Enwiki	4	2	43.50	248.00	22.40	37.90	158.00	13.70	5.59	9.13	8.77
Enwiki	4	3	34.80	179.00	16.20	26.70	139.00	19.00	8.08	4.01	-2.88
Enwiki	4	4	27.70	142.00	13.90	21.00	94.40	12.10	6.74	4.81	1.84
Enwiki	16	1	69.00	184.00	31.30	65.40	153.00	27.80	3.66	3.10	3.56
Enwiki	16	2	38.50	108.00	21.30	35.30	74.30	12.10	3.23	3.40	9.24
Enwiki	16	3	29.70	86.40	19.50	23.40	71.70	18.50	6.31	1.49	0.98
Enwiki	16	4	24.40	67.50	14.80	19.30	46.30	10.40	5.08	2.14	4.43
Enwiki	32	1	66.80	156.00	32.50	64.10	123.00	26.50	2.68	3.37	6.05
Enwiki	32	2	37.80	91.90	21.70	33.80	72.20	17.90	4.10	1.98	3.81
Enwiki	32	3	28.10	71.90	19.00	23.70	52.50	14.70	4.47	1.95	4.37
Enwiki	32	4	24.00	52.80	13.70	19.00	38.70	10.90	5.05	1.42	2.82
Ruwiki	4	1	133.00	766.00	33.80	114.00	501.00	34.50	10.50	15.00	-0.37
Ruwiki	4	2	69.20	424.00	24.80	59.70	292.00	23.40	5.40	7.51	0.80
Ruwiki	4	3	53.40	303.00	21.20	41.50	206.00	18.40	6.73	5.49	1.56
Ruwiki	4	4	43.30	246.00	17.60	32.10	171.00	18.30	6.34	4.28	-0.38
Ruwiki	16	1	110.00	291.00	35.30	105.00	256.00	34.80	3.05	2.01	0.29
Ruwiki	16	2	61.20	176.00	27.90	53.70	156.00	29.00	4.23	1.16	-0.62
Ruwiki	16	3	44.20	132.00	23.80	36.50	118.00	26.20	4.37	0.80	-1.37
Ruwiki	16	4	36.30	104.00	18.40	28.10	97.90	24.40	4.62	0.33	-3.37
Ruwiki	32	1	106.00	232.00	37.30	103.00	194.00	33.30	1.80	2.18	2.28
Ruwiki	32	2	59.40	139.00	27.10	54.00	106.00	19.70	3.07	1.89	4.19
Ruwiki	32	3	45.50	99.40	20.90	37.40	75.50	16.40	4.61	1.36	2.55
Ruwiki	32	4	35.10	85.60	20.60	28.90	68.60	18.40	3.55	0.96	1.28
Zhwiki	4	1	74.20	331.00	16.80	66.70	236.00	15.10	11.00	13.80	2.50
Zhwiki	4	2	40.60	182.00	12.80	34.80	132.00	11.90	8.48	7.35	1.34
Zhwiki	4	3	30.90	125.00	8.99	25.20	92.30	9.80	8.37	4.86	-1.19
Zhwiki	4	4	27.00	100.00	7.96	21.90	74.50	9.51	7.51	3.78	-2.27
Zhwiki	16	1	66.90	150.00	25.10	63.40	112.00	15.30	5.21	5.70	14.40
Zhwiki	16	2	36.50	77.50	12.80	32.50	62.20	11.90	5.96	2.24	1.37
Zhwiki	16	3	26.70	54.10	9.42	23.60	40.10	7.20	4.48	2.04	3.26
Zhwiki	16	4	23.80	44.20	8.49	20.70	32.20	6.82	4.59	1.76	2.46
Zhwiki	32	1	63.60	104.00	18.20	62.30	82.10	14.70	1.84	3.14	5.08
Zhwiki	32	2	35.50	56.90	10.10	32.50	41.00	7.57	4.46	2.33	3.64
Zhwiki	32	3	27.00	41.80	8.97	22.80	34.30	8.70	6.15	1.10	0.39
Zhwiki	32	4	23.10	35.50	8.57	20.20	25.90	7.02	4.21	1.42	2.27
								μ	**5.44**	3.94	2.12
								σ	2.36	3.87	3.12

Table D.117: `icgrep` PAPI comparison between `2020` and `CURR` on `SSE-4.2` with arguments Expression=\X, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	46.50	4310.00	122.00	34.40	2390.00	340.00	8.42	134.00	-152.00
Arwiki	4	2	24.70	2340.00	115.00	20.20	1380.00	190.00	3.13	67.10	-52.40
Arwiki	4	3	17.90	1620.00	125.00	15.50	936.00	129.00	1.63	48.10	-2.99
Arwiki	4	4	14.90	1240.00	116.00	12.50	747.00	123.00	1.68	34.20	-5.05
Arwiki	16	1	35.20	2440.00	241.00	31.30	1560.00	179.00	2.69	61.60	43.30
Arwiki	16	2	19.60	1270.00	219.00	17.70	827.00	113.00	1.29	31.10	74.10
Arwiki	16	3	14.40	867.00	181.00	13.00	587.00	95.90	0.97	19.60	59.60
Arwiki	16	4	12.10	657.00	150.00	11.50	430.00	64.20	0.43	15.80	59.70
Arwiki	32	1	33.20	2080.00	389.00	30.90	1460.00	182.00	1.63	42.80	144.00
Arwiki	32	2	19.10	1070.00	267.00	17.60	771.00	109.00	0.99	20.90	110.00
Arwiki	32	3	13.30	768.00	237.00	12.90	540.00	102.00	0.29	15.80	93.80
Arwiki	32	4	11.40	567.00	195.00	11.00	407.00	94.30	0.28	11.20	70.30
Enwiki	4	1	33.80	3040.00	102.00	24.40	1620.00	122.00	9.45	144.00	-19.40
Enwiki	4	2	18.40	1650.00	84.20	15.10	935.00	75.60	3.35	72.00	8.67
Enwiki	4	3	14.00	1140.00	87.60	11.40	669.00	74.50	2.67	47.80	13.20
Enwiki	4	4	11.20	878.00	93.80	9.91	501.00	55.80	1.35	38.00	38.40
Enwiki	16	1	25.70	1740.00	178.00	23.00	1100.00	146.00	2.76	64.00	33.20
Enwiki	16	2	14.70	895.00	160.00	13.70	568.00	76.80	1.02	32.90	83.50
Enwiki	16	3	11.20	609.00	129.00	9.97	425.00	78.70	1.29	18.50	50.50
Enwiki	16	4	9.11	475.00	121.00	8.61	316.00	57.90	0.50	16.10	63.30
Enwiki	32	1	24.20	1540.00	306.00	22.70	1060.00	148.00	1.60	48.10	159.00
Enwiki	32	2	14.10	775.00	205.00	13.00	578.00	99.90	1.07	19.80	106.00
Enwiki	32	3	10.40	544.00	169.00	10.50	369.00	61.00	-0.08	17.70	109.00
Enwiki	32	4	8.91	409.00	145.00	8.81	283.00	60.70	0.10	12.70	84.90
Ruwiki	4	1	53.60	5250.00	127.00	39.10	2820.00	392.00	8.21	137.00	-150.00
Ruwiki	4	2	28.40	2820.00	115.00	22.30	1690.00	236.00	3.45	63.80	-68.40
Ruwiki	4	3	20.60	1950.00	123.00	17.40	1140.00	148.00	1.81	46.00	-14.40
Ruwiki	4	4	16.60	1510.00	124.00	13.90	918.00	145.00	1.52	33.30	-11.70
Ruwiki	16	1	39.80	2890.00	237.00	35.10	1800.00	180.00	2.65	61.30	32.50
Ruwiki	16	2	21.30	1540.00	250.00	19.20	988.00	127.00	1.19	31.30	69.70
Ruwiki	16	3	15.60	1040.00	203.00	14.00	697.00	107.00	0.90	19.50	54.20
Ruwiki	16	4	13.00	790.00	170.00	11.50	553.00	98.20	0.85	13.40	40.90
Ruwiki	32	1	37.20	2410.00	407.00	34.50	1660.00	180.00	1.51	42.30	129.00
Ruwiki	32	2	20.40	1260.00	300.00	19.90	862.00	99.30	0.29	22.30	114.00
Ruwiki	32	3	15.40	849.00	232.00	14.80	580.00	82.80	0.29	15.20	84.30
Ruwiki	32	4	12.50	654.00	209.00	12.00	462.00	97.40	0.31	10.90	63.00
Zhwiki	4	1	28.70	2440.00	56.80	23.00	1510.00	142.00	8.37	137.00	-126.00
Zhwiki	4	2	15.20	1280.00	49.60	13.20	805.00	73.30	2.84	69.00	-34.70
Zhwiki	4	3	10.90	875.00	54.50	9.73	553.00	57.40	1.77	47.20	-4.18
Zhwiki	4	4	9.03	668.00	55.00	8.59	416.00	44.50	0.64	37.00	15.40
Zhwiki	16	1				21.10	823.00	80.90			
Zhwiki	16	2	12.40	651.00	102.00	11.50	438.00	53.60	1.28	31.30	70.40
Zhwiki	16	3	8.98	446.00	85.40	8.84	290.00	32.50	0.19	23.00	77.50
Zhwiki	16	4	7.77	334.00	69.00	7.67	224.00	30.10	0.14	16.00	57.10
Zhwiki	32	1	21.90	1030.00	177.00	20.80	734.00	81.60	1.64	43.80	141.00
Zhwiki	32	2	11.70	543.00	132.00	11.80	365.00	41.30	-0.18	26.10	134.00
Zhwiki	32	3	8.82	362.00	97.70	8.51	260.00	42.20	0.46	14.90	81.50
Zhwiki	32	4	7.50	280.00	87.60	7.53	195.00	38.40	-0.04	12.30	72.30
								μ	**1.88**	42.30	42.60
								σ	2.26	34.00	70.60

Table D.118: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX2` with arguments Expression=\X, Colours=always

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	28.90	367.00	33.30	25.10	246.00	113.00	2.64	8.46	-55.50
Arwiki	4	2	17.70	546.00	39.60	19.90	368.00	66.70	-1.52	12.40	-18.90
Arwiki	4	3	12.50	408.00	41.80	14.30	301.00	76.00	-1.30	7.47	-23.80
Arwiki	4	4	15.50	348.00	85.90	14.10	225.00	47.90	0.97	8.61	26.50
Arwiki	16	1	27.70	747.00	152.00	24.10	397.00	129.00	2.51	24.40	16.30
Arwiki	16	2	16.60	482.00	172.00	15.70	295.00	90.70	0.60	13.10	56.70
Arwiki	16	3	13.90	323.00	132.00	13.60	196.00	62.40	0.21	8.85	48.60
Arwiki	16	4	13.10	254.00	109.00	11.00	177.00	68.90	1.43	5.38	27.60
Arwiki	32	1	26.50	842.00	237.00	24.10	476.00	132.00	1.71	25.50	73.00
Arwiki	32	2	16.70	453.00	174.00	16.60	265.00	69.60	0.08	13.20	72.90
Arwiki	32	3	13.60	314.00	137.00	11.00	215.00	88.30	1.78	6.91	34.10
Arwiki	32	4	12.70	243.00	113.00	11.60	149.00	55.70	0.72	6.53	40.30
Enwiki	4	1	27.80	333.00	89.60	19.30	147.00	89.20	8.53	18.80	0.40
Enwiki	4	2	19.70	407.00	72.80	14.80	274.00	59.50	4.95	13.40	13.50
Enwiki	4	3	15.00	314.00	72.60	12.30	205.00	47.20	2.72	11.00	25.60
Enwiki	4	4	13.50	255.00	63.30	10.80	169.00	44.80	2.73	8.66	18.70
Enwiki	16	1	21.90	542.00	123.00	18.50	261.00	99.20	3.46	28.40	23.90
Enwiki	16	2	15.20	342.00	114.00	11.90	208.00	78.50	3.37	13.50	36.00
Enwiki	16	3	12.20	236.00	97.50	11.30	138.00	49.00	0.95	9.92	48.90
Enwiki	16	4	11.90	181.00	80.80	9.41	116.00	48.00	2.55	6.53	33.00
Enwiki	32	1	21.00	623.00	181.00	18.60	353.00	106.00	2.42	27.30	75.40
Enwiki	32	2	15.40	320.00	121.00	12.90	188.00	58.00	2.54	13.30	63.80
Enwiki	32	3	12.00	221.00	96.70	10.90	130.00	44.00	1.16	9.18	53.10
Enwiki	32	4	11.90	174.00	83.50	9.75	100.00	38.00	2.15	7.41	45.90
Ruwiki	4	1	34.40	386.00	36.50	27.40	236.00	124.00	3.96	8.50	-49.50
Ruwiki	4	2	26.80	677.00	113.00	19.80	449.00	93.50	3.97	12.90	11.00
Ruwiki	4	3	20.80	505.00	97.30	14.60	358.00	89.00	3.46	8.32	4.69
Ruwiki	4	4	17.70	427.00	92.40	14.40	271.00	57.30	1.91	8.81	19.90
Ruwiki	16	1	30.20	803.00	160.00	25.90	393.00	132.00	2.44	23.20	15.70
Ruwiki	16	2	20.00	521.00	157.00	15.90	333.00	105.00	2.33	10.60	29.20
Ruwiki	16	3	15.70	365.00	135.00	12.60	253.00	87.00	1.77	6.33	27.10
Ruwiki	16	4	14.20	285.00	112.00	10.80	210.00	81.30	1.94	4.26	17.60
Ruwiki	32	1	28.90	923.00	241.00	25.90	502.00	134.00	1.68	23.80	60.40
Ruwiki	32	2	18.40	500.00	178.00	17.10	300.00	79.50	0.79	11.30	56.00
Ruwiki	32	3	14.30	352.00	147.00	13.40	213.00	69.90	0.52	7.84	43.40
Ruwiki	32	4	13.70	269.00	118.00	12.30	163.00	57.30	0.81	6.00	34.60
Zhwiki	4	1	21.30	219.00	52.20	15.50	108.00	53.00	8.46	16.40	-1.26
Zhwiki	4	2	14.00	268.00	46.70	10.70	177.00	38.40	4.97	13.30	12.10
Zhwiki	4	3	9.94	206.00	46.90	8.41	133.00	29.50	2.24	10.80	25.50
Zhwiki	4	4	9.38	160.00	37.30	6.98	111.00	30.10	3.52	7.21	10.60
Zhwiki	16	1	16.90	375.00	72.50	14.80	178.00	58.40	3.01	28.90	20.70
Zhwiki	16	2	10.60	232.00	71.50	9.49	133.00	36.10	1.62	14.60	52.00
Zhwiki	16	3	8.28	155.00	56.50	6.90	103.00	36.60	2.02	7.60	29.20
Zhwiki	16	4	7.46	125.00	50.90	6.37	77.90	27.60	1.60	6.86	34.30
Zhwiki	32	1	16.30	414.00	110.00	14.80	227.00	60.20	2.21	27.40	73.30
Zhwiki	32	2	9.95	223.00	83.60	9.25	129.00	35.30	1.02	13.90	71.00
Zhwiki	32	3	7.93	147.00	58.80	7.44	87.50	26.70	0.73	8.76	47.10
Zhwiki	32	4	7.45	115.00	50.60	6.45	68.70	25.80	1.48	6.85	36.50
								μ	**2.20**	12.60	29.50
								σ	1.86	6.80	28.60

Table D.119: `icgrep` PAPI comparison between `2020` and `CURR` on `AVX-512` with arguments Expression=\X, Colours=always

D.4.9 Case study: `ztf-hash compression`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	32.40	151.00	2.51	29.00	177.00	2.75	3.01	-2.31	-0.20
Arwiki	4	2	17.10	115.00	1.10	14.80	129.00	1.04	2.02	-1.23	0.05
Arwiki	4	3	12.30	81.70	.72	9.97	90.10	.70	2.05	-0.73	0.02
Arwiki	4	4	10.20	61.30	.56	7.65	70.50	.52	2.21	-0.81	0.03
Arwiki	16	1	28.40	183.00	5.54	27.40	205.00	5.53	0.87	-2.01	0.01
Arwiki	16	2	14.80	99.20	2.70	13.80	112.00	2.83	0.85	-1.08	-0.11
Arwiki	16	3	10.60	66.20	1.73	9.17	71.30	1.89	1.25	-0.45	-0.14
Arwiki	16	4	9.33	48.40	1.20	7.02	55.40	1.44	2.02	-0.62	-0.21
Arwiki	32	1	27.40	182.00	5.49	27.00	187.00	5.52	0.33	-0.43	-0.03
Arwiki	32	2	14.90	96.00	2.65	13.50	101.00	2.80	1.25	-0.48	-0.13
Arwiki	32	3	9.95	64.50	1.77	8.99	65.20	2.05	0.84	-0.06	-0.25
Arwiki	32	4	8.82	49.40	1.27	6.85	48.90	2.29	1.73	0.04	-0.89
Enwiki	4	1	22.20	105.00	1.69	19.90	125.00	1.84	2.51	-2.11	-0.16
Enwiki	4	2	11.70	77.70	.66	10.30	92.30	.65	1.46	-1.55	0.02
Enwiki	4	3	7.89	56.90	.55	6.83	63.70	.53	1.13	-0.72	0.03
Enwiki	4	4	6.93	41.70	.36	5.33	48.50	.32	1.70	-0.72	0.05
Enwiki	16	1	19.50	126.00	3.79	19.00	143.00	3.78	0.46	-1.80	0.01
Enwiki	16	2	10.50	68.90	1.83	9.47	79.00	1.95	1.08	-1.08	-0.13
Enwiki	16	3	7.29	46.10	1.22	6.29	49.70	1.31	1.06	-0.38	-0.09
Enwiki	16	4	6.40	34.80	.88	4.85	38.90	.99	1.65	-0.43	-0.12
Enwiki	32	1	18.70	129.00	3.73	18.40	134.00	3.83	0.36	-0.58	-0.10
Enwiki	32	2	9.90	68.00	1.83	9.30	74.20	1.92	0.64	-0.66	-0.09
Enwiki	32	3	7.20	46.50	1.20	6.19	46.60	1.43	1.07	-0.01	-0.25
Enwiki	32	4	6.43	35.70	.88	4.87	35.50	1.72	1.67	0.02	-0.89
Ruwiki	4	1	36.70	195.00	3.08	32.00	228.00	3.35	2.94	-2.05	-0.16
Ruwiki	4	2	18.50	149.00	1.23	16.70	164.00	1.17	1.07	-0.96	0.04
Ruwiki	4	3	13.80	105.00	1.00	11.20	114.00	.94	1.60	-0.57	0.04
Ruwiki	4	4	11.40	77.60	.64	8.63	89.00	.59	1.73	-0.71	0.03
Ruwiki	16	1	31.70	230.00	6.80	30.50	257.00	6.82	0.76	-1.66	-0.01
Ruwiki	16	2	16.80	123.00	3.15	15.30	138.00	3.49	0.97	-0.91	-0.21
Ruwiki	16	3	11.60	84.30	2.11	10.20	88.90	2.33	0.92	-0.28	-0.13
Ruwiki	16	4	10.20	62.00	1.43	7.79	70.70	1.77	1.50	-0.54	-0.21
Ruwiki	32	1	30.40	227.00	6.73	29.80	235.00	6.79	0.39	-0.49	-0.04
Ruwiki	32	2	16.30	122.00	3.26	15.00	128.00	3.43	0.86	-0.37	-0.10
Ruwiki	32	3	11.50	82.60	2.17	9.97	81.70	2.55	0.98	0.05	-0.24
Ruwiki	32	4	10.10	63.00	1.55	7.58	62.00	2.91	1.58	0.07	-0.84
Zhwiki	4	1	16.10	76.40	1.22	14.50	92.40	1.26	2.49	-2.54	-0.06
Zhwiki	4	2	8.20	57.30	.47	7.51	63.80	.44	1.09	-1.03	0.04
Zhwiki	4	3	5.98	40.10	.40	5.09	45.30	.37	1.41	-0.82	0.05
Zhwiki	4	4	5.14	29.30	.26	3.96	34.70	.22	1.87	-0.85	0.07
Zhwiki	16	1	14.00	88.10	2.67	13.60	101.00	2.66	0.60	-2.01	0.01
Zhwiki	16	2	7.49	47.60	1.24	6.85	51.50	1.34	1.01	-0.61	-0.16
Zhwiki	16	3	5.16	32.60	.83	4.59	34.20	.90	0.90	-0.25	-0.10
Zhwiki	16	4	4.57	24.10	.60	3.53	26.60	.68	1.65	-0.40	-0.13
Zhwiki	32	1	13.40	89.10	2.58	13.40	91.70	2.66	0.01	-0.42	-0.14
Zhwiki	32	2	7.43	46.60	1.24	6.74	48.10	1.34	1.09	-0.24	-0.15
Zhwiki	32	3	4.92	31.10	.86	4.49	31.30	.99	0.68	-0.03	-0.20
Zhwiki	32	4	4.44	23.60	.62	3.45	23.00	1.24	1.56	0.08	-0.99
								μ	**1.31**	-0.79	-0.15
								σ	0.66	0.68	0.25

Table D.120: `ztf compression` PAPI comparison between `2020` and `CURR` on `SSE-4.2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	21.60	932.00	31.20	18.30	1010.00	30.10	2.90	-7.04	1.02
Arwiki	4	2	11.10	790.00	14.90	9.99	764.00	14.00	1.00	2.26	0.87
Arwiki	4	3	7.47	560.00	9.73	6.82	540.00	9.22	0.57	1.79	0.45
Arwiki	4	4	5.76	436.00	7.14	5.35	411.00	6.95	0.37	2.25	0.17
Arwiki	16	1	18.80	1060.00	30.70	17.60	1060.00	30.30	1.04	0.25	0.38
Arwiki	16	2	9.43	603.00	15.10	9.00	556.00	14.80	0.38	4.21	0.28
Arwiki	16	3	6.30	410.00	10.10	6.08	373.00	9.86	0.19	3.30	0.20
Arwiki	16	4	4.84	310.00	7.50	4.73	282.00	7.42	0.09	2.41	0.07
Arwiki	32	1	18.10	1080.00	30.60	17.40	1010.00	30.50	0.58	5.65	0.09
Arwiki	32	2	9.09	564.00	15.10	8.87	520.00	14.90	0.20	3.92	0.17
Arwiki	32	3	6.06	386.00	10.10	5.96	349.00	10.20	0.09	3.24	-0.07
Arwiki	32	4	4.66	290.00	7.53	4.61	265.00	10.10	0.04	2.24	-2.23
Enwiki	4	1	14.90	670.00	21.60	12.60	686.00	20.60	2.44	-1.67	1.05
Enwiki	4	2	7.69	547.00	10.40	6.94	534.00	9.71	0.80	1.43	0.71
Enwiki	4	3	5.15	388.00	6.79	4.69	377.00	6.43	0.49	1.20	0.38
Enwiki	4	4	3.98	304.00	4.99	3.75	285.00	4.82	0.24	2.05	0.17
Enwiki	16	1	13.00	740.00	21.30	12.20	721.00	21.00	0.88	2.02	0.34
Enwiki	16	2	6.54	422.00	10.60	6.23	383.00	10.30	0.32	4.19	0.31
Enwiki	16	3	4.36	287.00	7.01	4.20	255.00	6.85	0.17	3.37	0.16
Enwiki	16	4	3.43	217.00	5.24	3.40	194.00	5.15	0.04	2.42	0.09
Enwiki	32	1	12.50	766.00	21.20	12.00	680.00	21.20	0.51	9.09	0.05
Enwiki	32	2	6.33	403.00	10.60	6.14	357.00	10.40	0.20	4.88	0.18
Enwiki	32	3	4.20	273.00	6.98	4.12	238.00	7.09	0.08	3.73	-0.12
Enwiki	32	4	3.41	208.00	5.24	3.35	181.00	7.13	0.06	2.94	-2.00
Ruwiki	4	1	23.80	1160.00	38.50	19.80	1290.00	36.90	2.48	-8.28	0.99
Ruwiki	4	2	12.30	978.00	18.50	10.90	948.00	17.30	0.84	1.90	0.71
Ruwiki	4	3	8.31	691.00	12.00	7.46	666.00	11.40	0.53	1.52	0.36
Ruwiki	4	4	6.36	542.00	8.83	5.84	511.00	8.63	0.32	1.91	0.12
Ruwiki	16	1	20.30	1320.00	37.80	18.90	1310.00	37.40	0.91	1.05	0.24
Ruwiki	16	2	10.20	751.00	18.70	9.71	691.00	18.30	0.32	3.76	0.23
Ruwiki	16	3	6.81	510.00	12.40	6.54	459.00	12.20	0.17	3.20	0.17
Ruwiki	16	4	5.22	385.00	9.28	5.04	349.00	9.15	0.11	2.23	0.08
Ruwiki	32	1	19.50	1330.00	37.70	18.60	1240.00	37.70	0.51	5.72	-0.01
Ruwiki	32	2	9.81	706.00	18.60	9.53	644.00	18.50	0.17	3.82	0.10
Ruwiki	32	3	6.53	475.00	12.40	6.41	430.00	12.70	0.08	2.81	-0.16
Ruwiki	32	4	4.96	358.00	9.27	4.90	327.00	13.40	0.04	1.88	-2.59
Zhwiki	4	1	10.40	523.00	14.80	8.85	539.00	14.20	2.45	-2.56	0.96
Zhwiki	4	2	5.34	389.00	7.18	4.83	377.00	6.76	0.80	1.89	0.67
Zhwiki	4	3	3.62	273.00	4.69	3.32	264.00	4.46	0.48	1.47	0.36
Zhwiki	4	4	2.83	215.00	3.43	2.63	204.00	3.35	0.32	1.77	0.14
Zhwiki	16	1	9.03	530.00	14.70	8.49	514.00	14.40	0.86	2.43	0.43
Zhwiki	16	2	4.55	295.00	7.24	4.36	269.00	7.04	0.29	4.15	0.31
Zhwiki	16	3	3.06	201.00	4.83	2.96	181.00	4.71	0.16	3.17	0.20
Zhwiki	16	4	2.36	151.00	3.62	2.30	137.00	3.53	0.09	2.35	0.14
Zhwiki	32	1	8.70	530.00	14.60	8.40	477.00	14.60	0.48	8.38	0.06
Zhwiki	32	2	4.40	274.00	7.26	4.29	246.00	7.11	0.17	4.42	0.24
Zhwiki	32	3	2.95	186.00	4.84	2.90	165.00	4.85	0.08	3.25	-0.02
Zhwiki	32	4	2.31	142.00	3.64	2.23	125.00	4.69	0.13	2.62	-1.66
								μ	**0.55**	2.40	0.10
								σ	0.67	2.86	0.74

Table D.121: `ztf compression` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	16.90	23.90	21.90	14.30	24.60	22.40	2.24	-0.07	-0.43
Arwiki	4	2	10.50	309.00	11.60	9.60	318.00	11.70	0.80	-0.80	-0.15
Arwiki	4	3	7.28	233.00	7.66	6.76	240.00	7.93	0.45	-0.57	-0.23
Arwiki	4	4	5.83	192.00	5.76	5.87	200.00	5.99	-0.04	-0.69	-0.20
Arwiki	16	1	14.60	37.50	22.50	13.70	55.10	22.40	0.77	-1.55	0.03
Arwiki	16	2	8.14	165.00	12.90	8.03	198.00	13.80	0.09	-2.88	-0.81
Arwiki	16	3	5.52	119.00	8.90	5.61	144.00	10.00	-0.08	-2.21	-1.00
Arwiki	16	4	4.50	93.30	6.98	4.72	111.00	7.59	-0.19	-1.52	-0.54
Arwiki	32	1	14.60	129.00	22.50	14.30	265.00	22.50	0.21	-11.90	0.00
Arwiki	32	2	7.83	159.00	12.50	7.51	184.00	11.90	0.29	-2.24	0.53
Arwiki	32	3	5.30	109.00	8.44	5.18	125.00	8.24	0.10	-1.38	0.18
Arwiki	32	4	4.43	85.50	6.30	4.41	95.80	6.49	0.01	-0.90	-0.16
Enwiki	4	1	11.80	16.80	15.50	10.00	17.30	15.50	1.85	-0.05	0.03
Enwiki	4	2	7.28	211.00	7.98	6.80	224.00	8.02	0.51	-1.38	-0.04
Enwiki	4	3	5.02	161.00	5.30	4.74	168.00	5.43	0.30	-0.80	-0.13
Enwiki	4	4	4.09	133.00	4.00	4.23	142.00	4.19	-0.14	-0.94	-0.20
Enwiki	16	1	10.20	26.30	15.60	9.54	31.80	15.50	0.66	-0.59	0.04
Enwiki	16	2	5.65	112.00	8.77	5.59	136.00	9.42	0.07	-2.54	-0.69
Enwiki	16	3	3.86	81.70	6.11	3.94	98.40	6.89	-0.09	-1.76	-0.83
Enwiki	16	4	3.47	64.50	4.80	3.63	75.30	5.16	-0.18	-1.15	-0.38
Enwiki	32	1	10.10	85.60	15.60	10.10	190.00	15.60	0.07	-11.00	-0.01
Enwiki	32	2	5.43	113.00	9.13	5.24	128.00	8.26	0.20	-1.56	0.92
Enwiki	32	3	3.68	79.20	6.15	3.66	88.50	5.68	0.02	-0.98	0.50
Enwiki	32	4	3.46	61.20	4.54	3.48	66.60	4.51	-0.02	-0.58	0.03
Ruwiki	4	1	17.80	29.20	27.60	14.70	30.40	27.60	1.89	-0.07	0.01
Ruwiki	4	2	11.60	388.00	14.20	10.60	395.00	14.30	0.65	-0.40	-0.09
Ruwiki	4	3	8.06	294.00	9.43	7.47	297.00	9.73	0.36	-0.14	-0.18
Ruwiki	4	4	6.47	242.00	7.10	6.32	247.00	7.32	0.09	-0.30	-0.14
Ruwiki	16	1	14.90	50.10	27.70	14.00	79.20	27.70	0.59	-1.80	0.02
Ruwiki	16	2	8.65	203.00	15.70	8.54	244.00	16.50	0.07	-2.53	-0.48
Ruwiki	16	3	5.89	147.00	10.80	5.94	176.00	11.70	-0.03	-1.80	-0.56
Ruwiki	16	4	4.62	116.00	8.52	4.71	135.00	8.91	-0.06	-1.21	-0.24
Ruwiki	32	1	14.90	148.00	27.70	14.80	335.00	27.80	0.02	-11.50	-0.03
Ruwiki	32	2	8.15	200.00	16.30	7.82	227.00	14.70	0.20	-1.68	1.00
Ruwiki	32	3	5.53	139.00	10.90	5.36	154.00	10.10	0.11	-0.91	0.52
Ruwiki	32	4	4.31	108.00	8.04	4.23	118.00	8.01	0.05	-0.60	0.02
Zhwiki	4	1	7.79	11.70	10.70	6.76	11.90	10.70	1.63	-0.04	0.05
Zhwiki	4	2	4.87	143.00	5.49	4.60	151.00	5.52	0.43	-1.21	-0.04
Zhwiki	4	3	3.37	110.00	3.66	3.22	115.00	3.78	0.24	-0.74	-0.20
Zhwiki	4	4	2.71	90.10	2.76	2.72	95.00	2.84	-0.02	-0.78	-0.13
Zhwiki	16	1	6.63	19.10	10.70	6.37	25.20	10.70	0.41	-0.98	0.03
Zhwiki	16	2	3.74	77.20	6.06	3.81	94.20	6.43	-0.12	-2.70	-0.59
Zhwiki	16	3	2.57	56.50	4.19	2.65	68.00	4.51	-0.12	-1.83	-0.51
Zhwiki	16	4	2.05	44.90	3.32	2.12	52.30	3.39	-0.11	-1.18	-0.12
Zhwiki	32	1	6.60	61.50	10.70	6.71	133.00	10.70	-0.18	-11.30	0.03
Zhwiki	32	2	3.55	77.60	6.38	3.53	88.40	5.67	0.04	-1.72	1.13
Zhwiki	32	3	2.44	54.30	4.23	2.42	59.80	3.89	0.03	-0.88	0.53
Zhwiki	32	4	1.93	42.10	3.14	1.92	45.80	3.12	0.01	-0.59	0.02
								μ	**0.29**	-2.02	-0.07
								σ	0.55	2.93	0.43

Table D.122: `ztf compression` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.4.10 Case study: `ztf-hash decompression`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	60.30	380.00	3.31	57.10	320.00	3.09	2.74	5.25	0.20
Arwiki	4	2	33.30	201.00	3.82	29.60	190.00	1.44	3.26	0.96	2.09
Arwiki	4	3	22.80	142.00	7.44	20.20	131.00	1.02	2.28	0.92	5.64
Arwiki	4	4	19.70	112.00	8.66	15.00	98.70	.76	4.08	1.18	6.93
Arwiki	16	1	54.30	208.00	20.20	53.10	223.00	4.46	1.04	-1.30	13.80
Arwiki	16	2	29.20	117.00	15.70	26.90	118.00	2.21	2.03	-0.16	11.90
Arwiki	16	3	19.70	82.30	11.00	17.80	79.30	1.51	1.73	0.26	8.32
Arwiki	16	4	17.90	66.20	9.96	13.30	61.00	1.21	4.01	0.45	7.69
Arwiki	32	1				52.80	207.00	4.37			
Arwiki	32	2	29.30	116.00	16.40	26.00	109.00	2.37	2.87	0.65	12.40
Arwiki	32	3	19.50	82.70	13.20	17.40	70.10	2.81	1.85	1.11	9.09
Arwiki	32	4	18.70	68.50	11.80	13.20	50.00	4.40	4.85	1.63	6.51
Enwiki	4	1	45.00	287.00	2.10	42.10	239.00	2.10	3.05	5.11	-0.00
Enwiki	4	2	24.40	152.00	2.62	21.60	137.00	.99	2.98	1.54	1.73
Enwiki	4	3	18.00	107.00	4.80	14.30	92.30	.67	3.91	1.57	4.39
Enwiki	4	4	14.10	85.40	5.90	10.70	70.60	.51	3.61	1.57	5.72
Enwiki	16	1	40.40	155.00	14.20	39.10	166.00	3.71	1.41	-1.23	11.20
Enwiki	16	2	21.60	87.10	11.50	20.20	88.80	1.84	1.52	-0.17	10.30
Enwiki	16	3	16.10	61.40	8.37	13.10	56.80	1.24	3.21	0.49	7.57
Enwiki	16	4	12.70	48.60	7.00	9.84	44.30	.95	2.99	0.46	6.42
Enwiki	32	1	40.10	147.00	22.00	38.40	149.00	3.57	1.76	-0.24	19.50
Enwiki	32	2	22.20	81.10	12.00	19.30	79.60	1.85	3.11	0.15	10.70
Enwiki	32	3	15.30	58.10	9.30	12.80	49.00	2.06	2.64	0.96	7.69
Enwiki	32	4	14.60	48.30	8.22	9.71	35.20	3.20	5.19	1.39	5.32
Ruwiki	4	1	75.50	504.00	3.72	70.30	429.00	3.79	3.22	4.62	-0.04
Ruwiki	4	2	40.30	272.00	5.04	36.50	244.00	1.75	2.33	1.73	2.03
Ruwiki	4	3	28.20	190.00	9.70	24.40	169.00	1.19	2.40	1.31	5.27
Ruwiki	4	4	25.80	149.00	11.20	18.20	128.00	.88	4.75	1.32	6.39
Ruwiki	16	1	67.10	273.00	25.60	64.60	296.00	6.33	1.54	-1.45	11.90
Ruwiki	16	2	36.40	151.00	17.60	33.30	157.00	3.17	1.91	-0.39	8.92
Ruwiki	16	3	26.00	109.00	14.00	21.70	104.00	2.10	2.64	0.30	7.37
Ruwiki	16	4	22.00	86.30	12.30	16.40	80.40	1.68	3.46	0.37	6.55
Ruwiki	32	1	66.30	257.00	34.70	63.90	261.00	6.11	1.46	-0.29	17.70
Ruwiki	32	2	35.20	142.00	20.40	32.00	141.00	3.20	1.95	0.08	10.60
Ruwiki	32	3	25.80	102.00	15.80	21.30	87.90	3.62	2.79	0.89	7.52
Ruwiki	32	4	22.80	83.80	14.70	16.20	62.50	5.49	4.14	1.32	5.69
Zhwiki	4	1	31.70	200.00	1.40	29.20	169.00	1.45	3.93	4.84	-0.08
Zhwiki	4	2	16.40	107.00	1.72	14.90	94.20	.69	2.36	2.06	1.63
Zhwiki	4	3	12.10	73.30	3.34	10.10	64.30	.47	3.25	1.43	4.56
Zhwiki	4	4	10.90	57.20	3.73	7.58	48.90	.35	5.20	1.32	5.36
Zhwiki	16	1	28.10	105.00	9.69	27.30	114.00	2.48	1.27	-1.39	11.40
Zhwiki	16	2	15.60	58.10	6.56	13.80	59.50	1.23	2.82	-0.23	8.45
Zhwiki	16	3	11.00	41.10	4.94	9.09	39.60	.82	3.02	0.23	6.54
Zhwiki	16	4	10.20	32.50	4.20	6.86	30.10	.66	5.35	0.39	5.61
Zhwiki	32	1	27.80	99.60	13.50	26.70	101.00	2.39	1.81	-0.25	17.60
Zhwiki	32	2	14.90	54.40	7.47	13.40	52.90	1.23	2.48	0.23	9.90
Zhwiki	32	3	11.40	38.80	5.60	8.93	33.50	1.48	3.88	0.85	6.54
Zhwiki	32	4	9.50	31.10	5.04	6.79	23.90	2.10	4.29	1.16	4.66
								μ	**2.94**	0.91	7.39
								σ	1.11	1.49	4.45

Table D.123: `ztf decompression` PAPI comparison between 2020 and CURR on SSE-4.2

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	25.90	2340.00	23.60	21.50	1670.00	23.20	3.80	59.00	0.34
Arwiki	4	2	13.90	1330.00	58.70	12.00	996.00	11.30	1.73	29.50	41.70
Arwiki	4	3	9.81	923.00	122.00	8.48	681.00	7.40	1.16	21.20	100.00
Arwiki	4	4	7.75	699.00	123.00	6.74	515.00	5.56	0.89	16.10	103.00
Arwiki	16	1	21.80	1620.00	225.00	20.40	1320.00	24.20	1.21	26.30	176.00
Arwiki	16	2	11.30	848.00	268.00	10.50	675.00	11.90	0.76	15.20	225.00
Arwiki	16	3	7.70	563.00	198.00	7.11	454.00	8.11	0.52	9.54	167.00
Arwiki	16	4	5.91	426.00	157.00	5.43	340.00	8.20	0.42	7.60	131.00
Arwiki	32	1	21.60	1570.00	510.00	20.10	1280.00	24.50	1.27	25.70	426.00
Arwiki	32	2	11.00	805.00	308.00	10.20	648.00	13.00	0.69	13.80	259.00
Arwiki	32	3	7.41	525.00	220.00	6.85	433.00	13.20	0.49	8.08	181.00
Arwiki	32	4	5.70	398.00	172.00	5.23	327.00	25.90	0.42	6.23	128.00
Enwiki	4	1	20.00	1750.00	19.50	16.40	1220.00	19.20	3.74	56.10	0.28
Enwiki	4	2	10.50	1010.00	58.60	8.77	732.00	9.31	1.85	29.40	52.40
Enwiki	4	3	7.25	699.00	87.00	6.05	496.00	6.12	1.27	21.50	85.90
Enwiki	4	4	5.65	530.00	85.70	4.75	377.00	4.61	0.96	16.20	86.10
Enwiki	16	1	16.80	1180.00	165.00	15.70	937.00	20.00	1.16	25.70	154.00
Enwiki	16	2	8.63	618.00	185.00	7.98	476.00	9.86	0.69	15.10	185.00
Enwiki	16	3	5.83	412.00	136.00	5.40	321.00	7.10	0.46	9.68	137.00
Enwiki	16	4	4.55	314.00	108.00	4.14	242.00	7.65	0.44	7.58	107.00
Enwiki	32	1	16.50	1090.00	348.00	15.50	884.00	20.20	1.07	22.30	348.00
Enwiki	32	2	8.36	562.00	212.00	7.85	456.00	10.10	0.54	11.30	214.00
Enwiki	32	3	5.62	367.00	153.00	5.28	306.00	10.20	0.36	6.54	151.00
Enwiki	32	4	4.33	280.00	119.00	4.05	231.00	22.30	0.30	5.13	103.00
Ruwiki	4	1	31.30	3060.00	34.80	25.30	2170.00	32.90	3.73	55.40	1.14
Ruwiki	4	2	16.60	1760.00	98.10	13.80	1300.00	15.90	1.77	28.40	50.90
Ruwiki	4	3	11.60	1230.00	156.00	9.63	890.00	10.50	1.21	20.90	90.10
Ruwiki	4	4	9.06	937.00	154.00	7.54	678.00	7.88	0.94	16.00	90.50
Ruwiki	16	1	25.70	2070.00	283.00	23.90	1660.00	34.20	1.13	25.30	154.00
Ruwiki	16	2	13.30	1090.00	330.00	12.20	848.00	16.90	0.72	15.30	194.00
Ruwiki	16	3	9.08	735.00	245.00	8.28	575.00	11.50	0.50	9.94	144.00
Ruwiki	16	4	6.96	554.00	195.00	6.36	436.00	12.20	0.37	7.36	113.00
Ruwiki	32	1	25.30	1930.00	622.00	23.60	1570.00	34.40	1.06	22.60	364.00
Ruwiki	32	2	12.90	998.00	380.00	12.00	808.00	17.40	0.56	11.80	225.00
Ruwiki	32	3	8.72	664.00	274.00	8.06	541.00	18.90	0.41	7.59	158.00
Ruwiki	32	4	6.69	497.00	215.00	6.13	407.00	36.70	0.35	5.60	111.00
Zhwiki	4	1	13.40	1230.00	13.10	11.00	879.00	12.90	3.69	55.50	0.29
Zhwiki	4	2	7.09	703.00	35.50	5.97	519.00	6.22	1.77	29.10	46.50
Zhwiki	4	3	4.92	486.00	60.90	4.15	352.00	4.10	1.21	21.30	90.10
Zhwiki	4	4	3.85	371.00	59.50	3.27	267.00	3.08	0.92	16.50	89.50
Zhwiki	16	1	11.20	812.00	112.00	10.50	649.00	13.40	1.10	25.80	156.00
Zhwiki	16	2	5.79	425.00	127.00	5.36	330.00	6.60	0.69	15.10	191.00
Zhwiki	16	3	3.95	285.00	93.60	3.65	223.00	4.51	0.49	9.81	141.00
Zhwiki	16	4	3.04	216.00	74.70	2.81	168.00	5.04	0.37	7.62	111.00
Zhwiki	32	1	11.00	747.00	240.00	10.40	607.00	13.50	1.04	22.20	359.00
Zhwiki	32	2	5.63	384.00	146.00	5.28	310.00	6.90	0.56	11.80	220.00
Zhwiki	32	3	3.81	252.00	105.00	3.56	207.00	7.38	0.40	7.14	155.00
Zhwiki	32	4	2.94	192.00	82.20	2.72	156.00	16.40	0.34	5.61	104.00
								$\boldsymbol{\mu}$	**1.07**	19.10	144.00
								$\boldsymbol{\sigma}$	0.90	13.50	92.70

Table D.124: `ztf decompression` PAPI comparison between `2020` and `CURR` on `AVX2`

CONFIG			2020			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	20.70	425.00	27.30	15.90	177.00	17.80	4.24	21.80	8.27
Arwiki	4	2	13.40	478.00	98.10	10.90	367.00	11.40	2.21	9.71	76.10
Arwiki	4	3	10.00	343.00	91.50	8.14	278.00	8.23	1.65	5.66	73.10
Arwiki	4	4	8.82	272.00	77.30	6.72	224.00	6.57	1.84	4.23	62.10
Arwiki	16	1	17.40	537.00	147.00	15.70	349.00	17.90	1.48	16.50	114.00
Arwiki	16	2	10.40	337.00	139.00	8.72	249.00	10.30	1.46	7.65	113.00
Arwiki	16	3	8.60	227.00	97.70	6.11	170.00	7.26	2.19	5.05	79.40
Arwiki	16	4	8.32	171.00	74.90	4.79	129.00	6.23	3.10	3.67	60.30
Arwiki	32	1	17.80	596.00	243.00	16.40	429.00	18.00	1.26	14.60	197.00
Arwiki	32	2	10.20	313.00	147.00	8.23	233.00	11.80	1.77	7.09	119.00
Arwiki	32	3	8.70	209.00	102.00	5.63	156.00	16.00	2.69	4.68	75.30
Arwiki	32	4	8.52	157.00	78.20	4.41	118.00	22.20	3.61	3.43	49.20
Enwiki	4	1	16.10	294.00	21.90	12.40	94.90	14.80	3.91	21.10	7.61
Enwiki	4	2	10.30	361.00	68.40	8.17	284.00	9.86	2.21	8.16	62.20
Enwiki	4	3	7.35	256.00	60.70	5.74	211.00	7.23	1.71	4.82	56.80
Enwiki	4	4	6.30	204.00	53.60	4.61	170.00	5.79	1.79	3.62	50.70
Enwiki	16	1	13.70	366.00	104.00	12.30	237.00	14.80	1.42	13.70	95.10
Enwiki	16	2	7.90	244.00	95.90	6.67	179.00	8.66	1.31	6.92	92.70
Enwiki	16	3	6.11	164.00	68.20	4.56	121.00	6.19	1.65	4.58	65.90
Enwiki	16	4	5.83	124.00	52.30	3.60	92.40	5.37	2.36	3.39	49.80
Enwiki	32	1	13.90	424.00	168.00	12.70	294.00	14.80	1.28	13.90	162.00
Enwiki	32	2	7.73	228.00	104.00	6.39	163.00	9.38	1.42	6.92	100.00
Enwiki	32	3	6.31	152.00	72.10	4.36	109.00	12.40	2.06	4.58	63.40
Enwiki	32	4	6.04	114.00	55.40	3.44	82.30	16.50	2.76	3.32	41.20
Ruwiki	4	1	25.40	528.00	35.30	18.30	165.00	25.30	4.41	22.50	6.24
Ruwiki	4	2	16.30	633.00	131.00	12.80	491.00	16.40	2.18	8.84	70.90
Ruwiki	4	3	12.00	450.00	121.00	9.22	368.00	11.80	1.73	5.09	67.90
Ruwiki	4	4	10.80	359.00	101.00	7.51	298.00	9.35	2.03	3.80	57.10
Ruwiki	16	1	20.20	647.00	180.00	18.10	420.00	25.30	1.34	14.10	95.90
Ruwiki	16	2	12.40	431.00	173.00	10.10	318.00	14.90	1.39	7.03	97.70
Ruwiki	16	3	10.30	291.00	122.00	6.95	216.00	10.60	2.09	4.61	68.90
Ruwiki	16	4	10.20	220.00	93.20	5.43	166.00	8.79	2.94	3.33	52.30
Ruwiki	32	1	20.80	745.00	300.00	18.80	527.00	25.40	1.22	13.50	170.00
Ruwiki	32	2	12.10	401.00	185.00	9.52	289.00	16.10	1.60	6.97	104.00
Ruwiki	32	3	10.70	267.00	128.00	6.51	195.00	21.30	2.59	4.51	65.90
Ruwiki	32	4	10.50	200.00	97.30	5.14	148.00	28.60	3.34	3.25	42.50
Zhwiki	4	1	10.70	203.00	13.80	8.14	66.80	9.86	4.12	21.60	6.19
Zhwiki	4	2	6.80	245.00	39.70	5.48	191.00	6.52	2.09	8.61	52.60
Zhwiki	4	3	4.91	175.00	38.60	3.92	144.00	4.73	1.58	5.00	53.70
Zhwiki	4	4	4.23	140.00	34.70	3.16	116.00	3.74	1.69	3.84	49.20
Zhwiki	16	1	8.77	250.00	60.70	8.00	161.00	9.87	1.21	14.10	80.60
Zhwiki	16	2	5.16	165.00	65.90	4.42	122.00	5.86	1.18	6.97	95.30
Zhwiki	16	3	4.12	112.00	46.80	3.05	82.80	4.16	1.71	4.59	67.70
Zhwiki	16	4	3.98	84.70	35.90	2.38	63.30	3.50	2.54	3.39	51.40
Zhwiki	32	1	8.97	291.00	113.00	8.24	200.00	9.90	1.16	14.40	164.00
Zhwiki	32	2	5.04	156.00	70.30	4.20	110.00	6.65	1.33	7.24	101.00
Zhwiki	32	3	4.21	104.00	49.20	2.88	74.50	8.40	2.11	4.68	64.70
Zhwiki	32	4	4.12	77.90	37.60	2.26	56.30	11.00	2.94	3.42	42.10
								$\boldsymbol{\mu}$	**2.12**	8.21	75.10
								$\boldsymbol{\sigma}$	0.86	5.50	40.00

Table D.125: `ztf decompression` PAPI comparison between `2020` and `CURR` on `AVX-512`

D.5 Comparison against hybrid pipeline

D.5.1 Case study: `icgrep -colours=never`

Arguments: @ -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.84	12.40	1.18	1.33	10.50	.03	-0.34	0.14	0.80
Arwiki	4	3	.92	11.20	1.27	1.06	9.60	.03	-0.10	0.11	0.87
Arwiki	4	4	.88	7.98	.69	1.18	8.57	.03	-0.21	-0.04	0.46
Arwiki	16	2	.71	7.58	1.60	1.16	3.74	.03	-0.32	0.27	1.10
Arwiki	16	3	.78	6.08	1.37	.83	2.86	.03	-0.04	0.23	0.94
Arwiki	16	4	.74	4.00	.83	.84	3.30	.03	-0.07	0.05	0.56
Arwiki	32	2	.71	6.90	1.66	1.15	2.00	.03	-0.31	0.34	1.14
Arwiki	32	3	.74	4.58	1.59	.79	1.54	.03	-0.03	0.21	1.09
Arwiki	32	4	.74	3.30	.89	.81	1.72	.03	-0.05	0.11	0.60
Enwiki	4	2	.58	8.08	.80	.90	7.20	.01	-0.32	0.09	0.79
Enwiki	4	3	.62	7.25	.82	.73	6.72	.02	-0.11	0.05	0.81
Enwiki	4	4	.59	5.11	.46	.74	6.91	.02	-0.14	-0.18	0.45
Enwiki	16	2	.49	4.58	.97	.80	2.83	.01	-0.31	0.18	0.96
Enwiki	16	3	.51	4.03	.86	.56	2.17	.01	-0.06	0.19	0.86
Enwiki	16	4	.47	2.50	.52	.57	2.40	.01	-0.11	0.01	0.51
Enwiki	32	2	.47	4.11	1.06	.78	1.28	.01	-0.31	0.28	1.05
Enwiki	32	3	.47	3.13	.89	.54	1.03	.02	-0.06	0.21	0.89
Enwiki	32	4	.49	2.14	.59	.55	1.20	.01	-0.06	0.09	0.58
Ruwiki	4	2	1.04	15.40	1.52	1.64	13.00	.04	-0.34	0.14	0.83
Ruwiki	4	3	1.14	13.60	1.63	1.32	12.00	.05	-0.10	0.09	0.90
Ruwiki	4	4	1.10	9.85	.83	1.33	12.20	.05	-0.13	-0.13	0.44
Ruwiki	16	2	.87	9.56	2.18	1.43	4.61	.04	-0.32	0.28	1.21
Ruwiki	16	3	.95	7.55	1.78	1.02	3.58	.05	-0.04	0.23	0.98
Ruwiki	16	4	.90	4.83	1.05	1.04	4.18	.04	-0.08	0.04	0.57
Ruwiki	32	2	.89	8.88	2.34	1.44	2.87	.04	-0.31	0.34	1.30
Ruwiki	32	3	.89	5.59	1.96	.97	2.01	.05	-0.04	0.20	1.08
Ruwiki	32	4	.91	4.17	1.13	1.00	2.30	.04	-0.05	0.11	0.61
Zhwiki	4	2	.40	5.61	.57	.62	4.95	.03	-0.32	0.10	0.79
Zhwiki	4	3	.42	5.05	.56	.51	4.59	.03	-0.13	0.07	0.78
Zhwiki	4	4	.40	3.68	.34	.52	4.67	.03	-0.17	-0.14	0.46
Zhwiki	16	2	.33	3.20	.68	.54	1.86	.03	-0.31	0.20	0.95
Zhwiki	16	3	.36	2.79	.62	.40	1.42	.03	-0.06	0.20	0.87
Zhwiki	16	4	.34	1.80	.40	.40	1.65	.03	-0.09	0.02	0.55
Zhwiki	32	2	.34	2.81	.73	.53	1.01	.03	-0.29	0.26	1.03
Zhwiki	32	3	.35	2.24	.70	.37	.79	.03	-0.03	0.21	0.99
Zhwiki	32	4	.34	1.55	.44	.38	.93	.03	-0.05	0.09	0.60
								μ	**-0.16**	0.13	0.82
								σ	0.12	0.12	0.23

Table D.126: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=@, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.62	61.40	12.10	.82	38.20	.03	-0.14	1.62	8.44
Arwiki	4	3	.64	43.30	7.95	.78	38.10	.03	-0.10	0.36	5.53
Arwiki	4	4	.64	33.30	5.92	.79	37.40	.03	-0.10	-0.29	4.11
Arwiki	16	2	.51	39.20	12.30	.70	15.70	.03	-0.13	1.64	8.58
Arwiki	16	3	.50	26.70	8.25	.54	18.20	.03	-0.03	0.59	5.74
Arwiki	16	4	.49	19.60	6.15	.56	17.20	.03	-0.05	0.17	4.27
Arwiki	32	2	.47	35.70	12.30	.69	13.80	.03	-0.15	1.53	8.59
Arwiki	32	3	.47	21.80	8.26	.51	13.80	.03	-0.02	0.56	5.75
Arwiki	32	4	.47	16.20	6.19	.52	14.40	.03	-0.04	0.12	4.30
Enwiki	4	2	.42	42.50	8.39	.57	26.50	.01	-0.15	1.61	8.45
Enwiki	4	3	.44	29.90	5.47	.54	26.70	.01	-0.10	0.32	5.50
Enwiki	4	4	.44	23.00	4.10	.55	26.10	.01	-0.11	-0.30	4.12
Enwiki	16	2	.35	27.10	8.50	.48	11.00	.01	-0.14	1.63	8.55
Enwiki	16	3	.35	18.50	5.69	.38	12.60	.01	-0.03	0.60	5.72
Enwiki	16	4	.35	13.60	4.26	.39	11.90	.01	-0.04	0.17	4.28
Enwiki	32	2	.34	24.70	8.49	.48	9.59	.01	-0.15	1.52	8.54
Enwiki	32	3	.33	15.10	5.71	.35	9.51	.01	-0.02	0.56	5.74
Enwiki	32	4	.33	11.20	4.29	.36	9.96	.01	-0.03	0.12	4.31
Ruwiki	4	2	.76	75.40	15.00	1.01	46.80	.05	-0.14	1.62	8.45
Ruwiki	4	3	.80	53.60	9.77	.97	46.90	.05	-0.10	0.38	5.50
Ruwiki	4	4	.78	41.10	7.31	.97	45.90	.06	-0.10	-0.27	4.11
Ruwiki	16	2	.62	48.20	15.10	.87	19.50	.05	-0.14	1.63	8.54
Ruwiki	16	3	.63	32.80	10.20	.68	22.60	.05	-0.03	0.58	5.73
Ruwiki	16	4	.61	24.10	7.56	.69	21.20	.06	-0.05	0.17	4.25
Ruwiki	32	2	.61	43.60	15.10	.85	17.40	.05	-0.14	1.49	8.54
Ruwiki	32	3	.59	27.00	10.20	.63	17.20	.05	-0.02	0.56	5.76
Ruwiki	32	4	.59	20.00	7.64	.64	17.70	.07	-0.03	0.13	4.29
Zhwiki	4	2	.29	28.90	5.80	.39	18.00	.03	-0.14	1.60	8.47
Zhwiki	4	3	.31	20.60	3.79	.37	17.70	.03	-0.10	0.42	5.51
Zhwiki	4	4	.31	15.90	2.85	.38	17.50	.04	-0.10	-0.23	4.12
Zhwiki	16	2	.24	18.70	5.88	.34	7.46	.03	-0.13	1.65	8.59
Zhwiki	16	3	.24	12.80	3.93	.26	8.62	.03	-0.03	0.61	5.72
Zhwiki	16	4	.24	9.38	2.94	.27	8.17	.04	-0.05	0.18	4.26
Zhwiki	32	2	.24	16.90	5.84	.33	6.59	.03	-0.14	1.52	8.52
Zhwiki	32	3	.23	10.40	3.93	.24	6.68	.03	-0.02	0.55	5.72
Zhwiki	32	4	.23	7.77	2.97	.25	6.96	.04	-0.03	0.12	4.30
								μ	**-0.08**	0.70	6.14
								σ	0.05	0.67	1.79

Table D.127: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=@, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.77	27.10	11.30	.90	20.10	.02	-0.10	0.49	7.85
Arwiki	4	3	.84	19.70	7.49	1.03	21.00	.02	-0.14	-0.09	5.22
Arwiki	4	4	.86	15.50	5.62	1.04	21.00	.02	-0.13	-0.38	3.91
Arwiki	16	2	.58	20.70	11.20	.70	9.70	.02	-0.08	0.77	7.82
Arwiki	16	3	.59	13.80	7.49	.65	10.50	.02	-0.04	0.23	5.21
Arwiki	16	4	.58	10.00	5.62	.64	10.20	.02	-0.04	-0.02	3.91
Arwiki	32	2	.53	18.40	11.20	.66	6.60	.02	-0.09	0.82	7.82
Arwiki	32	3	.54	11.50	7.49	.57	6.78	.02	-0.03	0.33	5.21
Arwiki	32	4	.54	8.43	5.62	.57	6.83	.02	-0.02	0.11	3.91
Enwiki	4	2	.53	18.60	7.84	.62	13.90	.00	-0.09	0.47	7.90
Enwiki	4	3	.56	13.80	5.20	.71	14.40	.00	-0.15	-0.06	5.23
Enwiki	4	4	.59	10.70	3.86	.72	14.10	.00	-0.13	-0.34	3.89
Enwiki	16	2	.40	14.00	7.80	.48	6.74	.00	-0.08	0.73	7.85
Enwiki	16	3	.40	9.38	5.19	.44	7.26	.00	-0.05	0.21	5.23
Enwiki	16	4	.41	6.98	3.90	.44	7.04	.00	-0.03	-0.01	3.92
Enwiki	32	2	.38	12.10	7.79	.46	4.55	.00	-0.07	0.76	7.85
Enwiki	32	3	.37	7.91	5.19	.39	4.66	.00	-0.03	0.33	5.23
Enwiki	32	4	.38	5.72	3.90	.39	4.69	.00	-0.01	0.10	3.92
Ruwiki	4	2	.96	33.50	13.90	1.10	24.90	.04	-0.08	0.48	7.83
Ruwiki	4	3	1.04	24.30	9.25	1.26	25.80	.04	-0.12	-0.09	5.22
Ruwiki	4	4	1.07	19.20	6.94	1.28	25.80	.05	-0.12	-0.38	3.91
Ruwiki	16	2	.71	25.60	13.80	.86	12.00	.04	-0.09	0.77	7.81
Ruwiki	16	3	.73	17.00	9.24	.78	13.00	.04	-0.03	0.22	5.21
Ruwiki	16	4	.73	12.30	6.94	.78	12.60	.05	-0.03	-0.02	3.90
Ruwiki	32	2	.64	22.70	13.80	.80	8.18	.04	-0.10	0.82	7.82
Ruwiki	32	3	.67	14.20	9.24	.69	8.41	.04	-0.02	0.33	5.21
Ruwiki	32	4	.70	10.40	6.93	.69	8.48	.04	0.00	0.11	3.90
Zhwiki	4	2	.37	12.80	5.40	.43	9.62	.02	-0.08	0.47	7.89
Zhwiki	4	3	.39	9.50	3.59	.49	9.87	.02	-0.15	-0.05	5.24
Zhwiki	4	4	.40	7.45	2.73	.50	9.90	.03	-0.14	-0.36	3.96
Zhwiki	16	2	.25	9.61	5.37	.33	4.69	.02	-0.12	0.72	7.85
Zhwiki	16	3	.28	6.47	3.59	.31	5.05	.02	-0.05	0.21	5.24
Zhwiki	16	4	.25	4.81	2.70	.30	4.91	.03	-0.08	-0.01	3.92
Zhwiki	32	2	.23	8.28	5.36	.31	3.20	.02	-0.12	0.75	7.84
Zhwiki	32	3	.25	5.48	3.59	.27	3.28	.02	-0.03	0.32	5.23
Zhwiki	32	4	.25	4.04	2.70	.27	3.31	.03	-0.03	0.11	3.92
									μ	**-0.07**	0.25	5.66
									σ	0.04	0.36	1.63

Table D.128: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=@, Colours=never

Arguments: Date -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	1.22	12.80	1.76	1.73	11.90	.62	-0.36	0.07	0.79
Arwiki	4	3	1.26	14.00	1.80	1.30	11.70	.61	-0.03	0.16	0.83
Arwiki	4	4	1.21	9.89	1.00	1.31	12.40	.61	-0.07	-0.18	0.28
Arwiki	16	2	1.02	8.77	2.43	1.60	5.62	.63	-0.41	0.22	1.26
Arwiki	16	3	1.07	8.20	2.11	1.07	5.02	.61	0.00	0.22	1.05
Arwiki	16	4	1.01	5.67	1.29	1.09	5.59	.60	-0.06	0.01	0.48
Arwiki	32	2	1.02	8.00	2.35	1.56	3.80	.62	-0.38	0.29	1.21
Arwiki	32	3	.99	7.47	2.17	1.03	3.37	.61	-0.02	0.29	1.09
Arwiki	32	4	.95	4.65	1.21	1.06	3.67	.61	-0.07	0.07	0.42
Enwiki	4	2	.74	8.02	.85	1.14	7.06	.01	-0.41	0.10	0.85
Enwiki	4	3	.76	7.62	.81	.84	7.16	.01	-0.09	0.05	0.81
Enwiki	4	4	.73	5.78	.46	.85	7.67	.01	-0.13	-0.19	0.45
Enwiki	16	2	.63	4.76	1.07	1.06	2.53	.01	-0.43	0.22	1.07
Enwiki	16	3	.63	4.53	.98	.69	2.15	.01	-0.06	0.24	0.97
Enwiki	16	4	.57	3.22	.52	.71	2.55	.01	-0.14	0.07	0.52
Enwiki	32	2	.62	4.27	1.10	1.06	1.39	.01	-0.44	0.29	1.09
Enwiki	32	3	.56	3.94	.99	.67	1.30	.01	-0.11	0.27	0.99
Enwiki	32	4	.55	2.67	.56	.69	1.27	.01	-0.14	0.14	0.55
Ruwiki	4	2	1.43	16.50	2.54	2.12	14.60	.72	-0.39	0.11	1.03
Ruwiki	4	3	1.61	16.40	2.06	1.60	14.50	.71	0.01	0.11	0.77
Ruwiki	4	4	1.48	12.60	1.48	1.62	15.30	.71	-0.07	-0.15	0.44
Ruwiki	16	2	1.28	10.80	3.01	1.95	7.13	.72	-0.38	0.21	1.29
Ruwiki	16	3	1.36	9.61	2.41	1.32	6.14	.70	0.03	0.20	0.96
Ruwiki	16	4	1.23	7.40	1.78	1.34	6.58	.70	-0.06	0.05	0.61
Ruwiki	32	2	1.22	10.60	3.33	1.91	4.47	.72	-0.39	0.35	1.48
Ruwiki	32	3	1.20	8.81	2.54	1.26	4.13	.71	-0.03	0.27	1.04
Ruwiki	32	4	1.11	6.07	1.75	1.30	4.53	.70	-0.10	0.09	0.59
Zhwiki	4	2	.57	6.40	1.00	.82	5.90	.35	-0.36	0.07	0.96
Zhwiki	4	3	.58	6.03	.90	.62	5.66	.34	-0.06	0.05	0.82
Zhwiki	4	4	.60	4.58	.54	.63	6.07	.34	-0.03	-0.22	0.29
Zhwiki	16	2	.51	4.04	1.12	.75	2.85	.35	-0.35	0.17	1.13
Zhwiki	16	3	.49	3.73	1.01	.51	2.56	.34	-0.03	0.17	0.98
Zhwiki	16	4	.47	2.75	.66	.52	2.86	.34	-0.07	-0.02	0.47
Zhwiki	32	2	.50	3.63	1.12	.74	2.07	.35	-0.35	0.23	1.13
Zhwiki	32	3	.44	3.61	1.08	.49	1.85	.34	-0.07	0.26	1.09
Zhwiki	32	4	.45	2.40	.68	.50	1.96	.34	-0.08	0.06	0.50
								μ	**-0.17**	0.12	0.84
								σ	0.16	0.14	0.30

Table D.129: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=Date, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.84	63.50	12.90	1.07	40.50	1.75	-0.16	1.61	7.80
Arwiki	4	3	.86	57.30	9.43	.95	39.80	1.77	-0.07	1.22	5.34
Arwiki	4	4	.79	42.20	6.80	.95	42.00	1.78	-0.12	0.01	3.50
Arwiki	16	2	.69	41.20	13.80	.96	20.00	1.74	-0.19	1.48	8.45
Arwiki	16	3	.69	34.90	9.55	.73	21.60	1.76	-0.02	0.93	5.44
Arwiki	16	4	.63	22.30	7.06	.75	20.90	1.78	-0.08	0.10	3.68
Arwiki	32	2	.66	37.30	13.30	.94	18.50	1.74	-0.20	1.31	8.04
Arwiki	32	3	.62	29.50	9.60	.69	18.20	1.76	-0.05	0.79	5.48
Arwiki	32	4	.61	19.10	6.89	.70	18.50	1.78	-0.07	0.04	3.56
Enwiki	4	2	.50	44.10	8.35	.67	26.70	.01	-0.17	1.76	8.41
Enwiki	4	3	.51	38.70	5.62	.60	26.90	.01	-0.09	1.19	5.65
Enwiki	4	4	.46	28.10	4.24	.60	27.80	.01	-0.14	0.04	4.26
Enwiki	16	2	.41	26.60	8.47	.62	11.00	.01	-0.21	1.58	8.53
Enwiki	16	3	.41	22.80	5.69	.45	12.50	.01	-0.04	1.03	5.73
Enwiki	16	4	.36	14.10	4.26	.46	11.70	.01	-0.10	0.24	4.29
Enwiki	32	2	.38	24.40	8.49	.61	9.75	.01	-0.23	1.47	8.55
Enwiki	32	3	.36	18.90	5.72	.43	9.56	.01	-0.07	0.94	5.76
Enwiki	32	4	.34	12.10	4.33	.45	9.89	.01	-0.10	0.22	4.35
Ruwiki	4	2	1.04	78.70	16.80	1.30	49.20	2.04	-0.14	1.67	8.34
Ruwiki	4	3	1.07	69.80	11.10	1.17	48.40	2.06	-0.06	1.21	5.13
Ruwiki	4	4	.96	53.10	9.11	1.18	51.20	2.08	-0.13	0.11	3.98
Ruwiki	16	2	.84	50.20	16.70	1.17	24.50	2.02	-0.18	1.46	8.31
Ruwiki	16	3	.87	42.00	11.10	.90	26.70	2.06	-0.01	0.87	5.15
Ruwiki	16	4	.76	28.10	9.13	.91	25.60	2.07	-0.08	0.14	4.00
Ruwiki	32	2	.79	47.30	17.00	1.15	22.80	2.03	-0.20	1.39	8.49
Ruwiki	32	3	.77	36.20	11.50	.85	22.40	2.05	-0.04	0.78	5.37
Ruwiki	32	4	.72	24.20	8.89	.86	22.90	2.08	-0.08	0.08	3.85
Zhwiki	4	2	.41	30.70	6.53	.52	19.60	.92	-0.16	1.63	8.23
Zhwiki	4	3	.42	27.40	4.62	.47	19.10	.94	-0.06	1.23	5.40
Zhwiki	4	4	.40	20.50	3.38	.47	20.10	.95	-0.11	0.06	3.56
Zhwiki	16	2	.34	19.50	6.41	.46	10.10	.91	-0.17	1.38	8.07
Zhwiki	16	3	.34	16.80	4.52	.36	10.90	.93	-0.02	0.86	5.27
Zhwiki	16	4	.32	10.90	3.40	.37	10.50	.94	-0.07	0.06	3.61
Zhwiki	32	2	.33	17.80	6.31	.45	9.62	.92	-0.18	1.20	7.92
Zhwiki	32	3	.30	14.90	4.79	.34	9.37	.92	-0.06	0.81	5.68
Zhwiki	32	4	.30	9.43	3.44	.34	9.51	.94	-0.07	-0.01	3.67
								μ	**-0.11**	0.86	5.86
								σ	0.06	0.60	1.84

Table D.130: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=Date, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.97	32.50	11.90	1.03	22.70	1.46	-0.04	0.68	7.29
Arwiki	4	3	1.02	22.90	8.55	1.13	23.20	1.47	-0.07	-0.03	4.94
Arwiki	4	4	1.07	17.10	6.17	1.14	23.00	1.49	-0.05	-0.41	3.27
Arwiki	16	2	.70	22.70	12.50	.82	12.30	1.45	-0.08	0.73	7.71
Arwiki	16	3	.73	15.60	8.58	.73	13.10	1.46	0.00	0.17	4.97
Arwiki	16	4	.73	11.50	6.38	.73	12.70	1.48	-0.00	-0.08	3.42
Arwiki	32	2	.68	19.40	12.00	.77	9.26	1.44	-0.06	0.71	7.35
Arwiki	32	3	.67	12.70	8.57	.65	9.43	1.45	0.01	0.23	4.97
Arwiki	32	4	.71	9.14	6.17	.66	9.47	1.47	0.04	-0.02	3.28
Enwiki	4	2	.57	21.30	7.79	.63	13.70	.00	-0.06	0.77	7.84
Enwiki	4	3	.58	14.90	5.19	.70	14.30	.00	-0.13	0.06	5.23
Enwiki	4	4	.60	11.30	3.88	.71	14.00	.00	-0.11	-0.28	3.91
Enwiki	16	2	.40	14.20	7.79	.50	6.64	.00	-0.10	0.76	7.84
Enwiki	16	3	.40	9.72	5.19	.44	7.28	.00	-0.03	0.25	5.23
Enwiki	16	4	.41	7.33	3.89	.44	7.05	.00	-0.03	0.03	3.92
Enwiki	32	2	.38	12.00	7.78	.48	4.55	.00	-0.10	0.75	7.84
Enwiki	32	3	.37	7.86	5.19	.38	4.68	.00	-0.02	0.32	5.23
Enwiki	32	4	.39	5.85	3.90	.39	4.71	.00	-0.01	0.12	3.92
Ruwiki	4	2	1.16	40.80	15.20	1.26	28.10	1.68	-0.06	0.72	7.68
Ruwiki	4	3	1.29	27.80	10.10	1.37	28.60	1.69	-0.04	-0.04	4.77
Ruwiki	4	4	1.28	21.80	8.20	1.39	28.40	1.71	-0.06	-0.37	3.67
Ruwiki	16	2	.88	27.60	15.10	.99	15.10	1.71	-0.06	0.70	7.58
Ruwiki	16	3	.93	18.70	10.00	.88	16.10	1.68	0.03	0.15	4.74
Ruwiki	16	4	.86	14.60	8.18	.89	15.60	1.70	-0.02	-0.06	3.67
Ruwiki	32	2	.78	24.70	15.40	.92	11.50	1.66	-0.08	0.75	7.77
Ruwiki	32	3	.82	15.50	10.30	.78	11.70	1.67	0.02	0.22	4.90
Ruwiki	32	4	.84	11.70	7.92	.80	11.70	1.68	0.03	-0.00	3.53
Zhwiki	4	2	.48	15.70	5.95	.50	11.20	.77	-0.03	0.66	7.60
Zhwiki	4	3	.49	11.40	4.19	.54	11.40	.78	-0.08	0.00	5.00
Zhwiki	4	4	.52	8.73	3.07	.55	11.30	.80	-0.05	-0.38	3.34
Zhwiki	16	2	.34	10.50	5.84	.39	6.23	.76	-0.07	0.62	7.45
Zhwiki	16	3	.36	7.64	4.11	.35	6.59	.78	0.01	0.15	4.90
Zhwiki	16	4	.34	5.60	3.06	.36	6.41	.79	-0.02	-0.12	3.34
Zhwiki	32	2	.32	8.82	5.73	.37	4.83	.75	-0.07	0.59	7.31
Zhwiki	32	3	.31	6.50	4.30	.31	4.90	.77	-0.01	0.23	5.19
Zhwiki	32	4	.34	4.57	3.07	.32	4.92	.77	0.03	-0.05	3.38
								μ	**-0.04**	0.24	5.39
								σ	0.04	0.37	1.69

Table D.131: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=Date, Colours=never

Arguments: Email -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.24	10.20	.39	3.67	9.81	.03	-1.00	0.03	0.26
Arwiki	4	3	3.19	26.70	.59	2.29	10.60	.03	0.63	1.13	0.39
Arwiki	4	4	3.46	27.60	.19	2.33	10.70	.03	0.79	1.18	0.11
Arwiki	16	2	2.10	6.31	.48	3.52	3.91	.03	-0.99	0.17	0.32
Arwiki	16	3	3.04	24.70	.62	2.10	3.79	.03	0.66	1.46	0.41
Arwiki	16	4	3.24	25.00	.25	2.16	3.95	.03	0.76	1.47	0.15
Arwiki	32	2	2.07	5.73	.54	3.51	1.94	.03	-1.01	0.26	0.36
Arwiki	32	3	3.05	24.00	.52	2.09	2.25	.03	0.67	1.52	0.34
Arwiki	32	4	3.30	25.30	.28	2.11	2.19	.03	0.83	1.61	0.18
Enwiki	4	2	1.03	7.90	1.11	1.78	6.76	.01	-0.75	0.11	1.11
Enwiki	4	3	.91	6.40	1.00	1.17	7.29	.01	-0.27	-0.09	0.99
Enwiki	4	4	.92	5.64	.54	1.32	7.13	.01	-0.41	-0.15	0.53
Enwiki	16	2	.94	4.87	1.29	1.68	2.67	.01	-0.75	0.22	1.29
Enwiki	16	3	.76	4.25	1.18	1.01	2.72	.01	-0.25	0.15	1.18
Enwiki	16	4	.77	3.64	.65	1.03	2.90	.01	-0.27	0.07	0.64
Enwiki	32	2	.92	4.12	1.40	1.67	1.77	.01	-0.76	0.24	1.39
Enwiki	32	3	.72	3.53	1.11	.99	1.73	.01	-0.27	0.18	1.10
Enwiki	32	4	.78	3.27	.71	1.01	1.64	.01	-0.23	0.16	0.70
Ruwiki	4	2	3.12	16.50	.49	4.80	12.20	.04	-0.95	0.24	0.25
Ruwiki	4	3	4.37	38.40	.73	3.10	13.00	.04	0.72	1.44	0.39
Ruwiki	4	4	4.72	40.50	.29	3.15	13.40	.04	0.89	1.54	0.14
Ruwiki	16	2	2.96	11.30	.56	4.62	4.55	.04	-0.94	0.38	0.29
Ruwiki	16	3	4.26	35.40	.79	2.94	4.76	.04	0.75	1.74	0.42
Ruwiki	16	4	4.49	37.40	.28	2.99	4.98	.04	0.85	1.84	0.13
Ruwiki	32	2	2.92	10.50	.60	4.60	2.65	.04	-0.95	0.44	0.32
Ruwiki	32	3	4.23	34.40	.70	2.92	2.88	.04	0.74	1.79	0.38
Ruwiki	32	4	4.55	37.00	.37	2.96	2.51	.04	0.90	1.95	0.19
Zhwiki	4	2	1.45	6.21	.22	2.16	4.66	.03	-1.05	0.23	0.28
Zhwiki	4	3	1.72	8.28	.30	1.56	5.08	.03	0.22	0.47	0.41
Zhwiki	4	4	1.90	7.85	.14	1.58	5.14	.03	0.47	0.40	0.16
Zhwiki	16	2	1.40	3.59	.24	2.09	1.88	.03	-1.02	0.25	0.32
Zhwiki	16	3	1.66	5.87	.35	1.51	1.88	.03	0.22	0.59	0.47
Zhwiki	16	4	1.83	5.05	.13	1.52	2.01	.03	0.45	0.45	0.15
Zhwiki	32	2	1.39	3.08	.25	2.09	1.20	.03	-1.03	0.28	0.33
Zhwiki	32	3	1.64	5.06	.30	1.50	1.20	.03	0.21	0.57	0.39
Zhwiki	32	4	1.79	4.54	.16	1.52	1.20	.03	0.40	0.49	0.19
								μ	**-0.05**	0.69	0.46
								σ	0.72	0.65	0.35

Table D.132: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=Email, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	1.49	66.10	17.00	2.03	42.50	.02	-0.38	1.65	11.80
Arwiki	4	3	2.09	83.90	11.50	1.44	40.30	.02	0.45	3.05	7.98
Arwiki	4	4	2.20	74.90	8.64	1.48	42.50	.03	0.50	2.26	6.01
Arwiki	16	2	1.32	48.80	16.90	1.92	17.00	.02	-0.42	2.22	11.80
Arwiki	16	3	1.90	69.20	11.30	1.24	20.60	.02	0.46	3.39	7.87
Arwiki	16	4	2.01	62.90	8.50	1.27	19.10	.03	0.51	3.06	5.91
Arwiki	32	2	1.28	41.20	16.90	1.92	14.90	.02	-0.45	1.83	11.80
Arwiki	32	3	1.86	65.00	11.30	1.23	15.90	.02	0.44	3.42	7.85
Arwiki	32	4	1.98	59.20	8.47	1.25	16.30	.03	0.51	2.99	5.89
Enwiki	4	2	.71	38.10	9.21	1.14	29.20	.01	-0.43	0.90	9.27
Enwiki	4	3	.69	36.20	6.11	.83	27.80	.01	-0.14	0.85	6.14
Enwiki	4	4	.73	30.80	4.54	.86	29.40	.01	-0.13	0.14	4.56
Enwiki	16	2	.59	24.10	9.11	1.05	11.90	.01	-0.46	1.23	9.17
Enwiki	16	3	.58	24.40	6.10	.66	13.70	.01	-0.08	1.08	6.14
Enwiki	16	4	.62	20.80	4.55	.68	12.70	.01	-0.06	0.82	4.57
Enwiki	32	2	.58	19.10	9.12	1.04	10.20	.01	-0.47	0.90	9.18
Enwiki	32	3	.57	21.80	6.11	.63	10.40	.01	-0.07	1.14	6.14
Enwiki	32	4	.60	18.80	4.57	.66	10.60	.01	-0.06	0.83	4.59
Ruwiki	4	2	2.19	93.40	21.10	2.76	51.50	.05	-0.32	2.38	11.90
Ruwiki	4	3	2.98	117.00	14.20	2.01	49.80	.05	0.55	3.78	8.03
Ruwiki	4	4	3.13	104.00	10.70	2.08	52.10	.06	0.59	2.92	6.02
Ruwiki	16	2	1.96	69.50	21.00	2.60	21.10	.05	-0.36	2.74	11.90
Ruwiki	16	3	2.73	97.40	14.00	1.85	26.00	.05	0.50	4.05	7.91
Ruwiki	16	4	2.89	90.40	10.50	1.86	24.80	.06	0.58	3.71	5.94
Ruwiki	32	2	1.91	62.70	21.00	2.58	18.20	.05	-0.38	2.52	11.90
Ruwiki	32	3	2.67	91.70	14.00	1.81	21.10	.05	0.49	4.00	7.91
Ruwiki	32	4	2.84	84.00	10.50	1.83	21.10	.06	0.57	3.57	5.92
Zhwiki	4	2	1.04	43.30	8.24	1.22	19.70	.03	-0.27	3.46	12.00
Zhwiki	4	3	1.30	51.00	5.50	.96	19.80	.03	0.50	4.59	8.02
Zhwiki	4	4	1.41	46.20	4.12	1.00	20.80	.04	0.60	3.73	5.99
Zhwiki	16	2	.95	33.90	8.19	1.17	7.94	.03	-0.32	3.82	12.00
Zhwiki	16	3	1.20	41.90	5.45	.89	10.80	.03	0.46	4.57	7.95
Zhwiki	16	4	1.29	40.20	4.08	.91	9.98	.04	0.56	4.43	5.94
Zhwiki	32	2	.94	30.50	8.19	1.18	7.12	.03	-0.35	3.43	12.00
Zhwiki	32	3	1.18	39.70	5.44	.87	8.53	.03	0.45	4.58	7.94
Zhwiki	32	4	1.28	38.10	4.08	.89	8.61	.04	0.57	4.32	5.93
								μ	**0.12**	2.73	8.11
								σ	0.42	1.30	2.49

Table D.133: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=Email, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	1.34	25.80	11.30	1.65	21.30	.02	-0.22	0.31	7.85
Arwiki	4	3	1.59	20.70	7.50	1.36	21.80	.02	0.16	-0.07	5.22
Arwiki	4	4	1.73	17.10	5.62	1.38	21.40	.02	0.24	-0.30	3.91
Arwiki	16	2	1.04	19.00	11.20	1.48	10.20	.02	-0.30	0.61	7.82
Arwiki	16	3	1.26	15.10	7.50	1.05	11.50	.02	0.15	0.25	5.22
Arwiki	16	4	1.38	12.60	5.61	1.08	10.60	.02	0.20	0.13	3.90
Arwiki	32	2	.98	15.50	11.20	1.45	6.98	.02	-0.32	0.59	7.81
Arwiki	32	3	1.21	13.00	7.48	1.02	7.25	.02	0.13	0.40	5.21
Arwiki	32	4	1.34	10.60	5.62	1.05	7.17	.02	0.20	0.24	3.91
Enwiki	4	2	.75	18.30	7.76	1.01	15.20	.00	-0.27	0.32	7.82
Enwiki	4	3	.77	14.30	5.20	.89	15.70	.00	-0.12	-0.14	5.23
Enwiki	4	4	.82	11.30	3.88	.90	15.40	.00	-0.08	-0.42	3.90
Enwiki	16	2	.58	12.20	7.80	.89	7.17	.00	-0.31	0.51	7.85
Enwiki	16	3	.61	9.33	5.20	.62	8.02	.00	-0.01	0.13	5.24
Enwiki	16	4	.66	7.48	3.90	.64	7.46	.00	0.02	0.00	3.93
Enwiki	32	2	.53	9.75	7.79	.86	4.88	.00	-0.33	0.49	7.85
Enwiki	32	3	.58	7.91	5.19	.57	5.03	.00	0.01	0.29	5.23
Enwiki	32	4	.63	6.18	3.90	.59	4.97	.00	0.04	0.12	3.93
Ruwiki	4	2	1.96	34.00	13.90	2.25	26.30	.04	-0.16	0.43	7.86
Ruwiki	4	3	2.35	27.20	9.25	1.84	26.60	.04	0.29	0.03	5.22
Ruwiki	4	4	2.59	22.60	6.91	1.88	26.30	.04	0.40	-0.21	3.89
Ruwiki	16	2	1.61	25.30	13.80	2.02	12.70	.04	-0.23	0.71	7.82
Ruwiki	16	3	1.96	20.40	9.25	1.55	14.10	.04	0.24	0.36	5.22
Ruwiki	16	4	2.20	17.30	6.92	1.59	13.20	.04	0.34	0.23	3.90
Ruwiki	32	2	1.54	21.00	13.90	1.97	8.68	.04	-0.24	0.70	7.83
Ruwiki	32	3	1.89	17.90	9.24	1.51	8.96	.04	0.22	0.51	5.21
Ruwiki	32	4	2.11	14.90	6.92	1.55	8.83	.04	0.32	0.34	3.90
Zhwiki	4	2	.86	13.10	5.36	.97	10.30	.02	-0.16	0.41	7.83
Zhwiki	4	3	.96	10.80	3.60	.80	10.30	.02	0.22	0.08	5.26
Zhwiki	4	4	1.04	8.89	2.72	.83	10.00	.03	0.31	-0.17	3.95
Zhwiki	16	2	.71	9.52	5.36	.89	4.92	.02	-0.25	0.68	7.84
Zhwiki	16	3	.85	7.76	3.60	.71	5.47	.02	0.20	0.34	5.26
Zhwiki	16	4	.88	6.33	2.70	.73	5.12	.03	0.21	0.18	3.91
Zhwiki	32	2	.70	7.98	5.36	.87	3.40	.02	-0.25	0.67	7.84
Zhwiki	32	3	.83	6.84	3.59	.70	3.50	.03	0.19	0.49	5.22
Zhwiki	32	4	.88	5.57	2.70	.72	3.46	.03	0.24	0.31	3.92
								μ	**0.03**	0.27	5.66
								σ	0.23	0.29	1.63

Table D.134: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=Email, Colours=never

Arguments: URIOrEmail -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.81	17.20	3.82	3.32	24.80	5.12	-0.36	-0.53	-0.91
Arwiki	4	3	2.62	16.10	3.02	2.45	23.40	5.06	0.12	-0.51	-1.42
Arwiki	4	4	2.69	16.80	3.06	2.46	24.10	5.03	0.16	-0.51	-1.38
Arwiki	16	2	2.58	11.30	3.81	3.12	18.00	5.12	-0.38	-0.47	-0.92
Arwiki	16	3	2.04	13.70	4.77	2.15	16.90	5.03	-0.08	-0.22	-0.18
Arwiki	16	4	2.51	10.90	2.73	2.15	18.00	5.04	0.25	-0.50	-1.61
Arwiki	32	2	2.41	12.80	5.15	3.14	17.20	5.13	-0.51	-0.31	0.01
Arwiki	32	3	2.09	11.70	4.03	2.08	15.80	5.09	0.01	-0.29	-0.74
Arwiki	32	4	2.38	9.70	2.94	2.09	15.90	5.03	0.20	-0.44	-1.46
Enwiki	4	2	1.15	6.71	.56	2.01	6.80	.01	-0.86	-0.01	0.56
Enwiki	4	3	1.04	6.72	.56	1.28	6.86	.01	-0.24	-0.01	0.55
Enwiki	4	4	1.07	5.34	.27	1.30	7.55	.01	-0.24	-0.22	0.26
Enwiki	16	2	1.05	3.24	.55	1.89	2.51	.01	-0.85	0.07	0.54
Enwiki	16	3	.88	3.37	.59	1.11	2.02	.01	-0.23	0.14	0.58
Enwiki	16	4	.94	2.85	.25	1.12	2.44	.01	-0.17	0.04	0.24
Enwiki	32	2	1.05	2.89	.61	1.92	1.29	.01	-0.88	0.16	0.60
Enwiki	32	3	.87	3.40	.68	1.09	1.09	.01	-0.22	0.23	0.67
Enwiki	32	4	.93	2.48	.27	1.11	1.04	.01	-0.18	0.14	0.27
Ruwiki	4	2	3.25	21.90	5.13	4.11	28.80	5.71	-0.48	-0.39	-0.33
Ruwiki	4	3	3.08	21.10	4.19	3.00	27.80	5.64	0.05	-0.38	-0.82
Ruwiki	4	4	3.33	20.90	3.50	3.01	28.30	5.63	0.18	-0.41	-1.21
Ruwiki	16	2	2.89	16.30	5.83	3.87	21.20	5.70	-0.56	-0.28	0.07
Ruwiki	16	3	2.77	13.20	4.06	2.59	19.80	5.63	0.10	-0.38	-0.89
Ruwiki	16	4	3.01	14.50	3.56	2.62	20.40	5.62	0.22	-0.34	-1.17
Ruwiki	32	2	3.03	13.10	4.97	3.86	18.80	5.72	-0.47	-0.32	-0.42
Ruwiki	32	3	2.65	12.40	4.06	2.54	18.00	5.65	0.07	-0.32	-0.90
Ruwiki	32	4	2.93	12.60	3.61	2.57	18.30	5.63	0.21	-0.32	-1.14
Zhwiki	4	2	1.24	11.10	3.10	1.67	12.20	2.63	-0.64	-0.17	0.69
Zhwiki	4	3	1.30	8.21	1.72	1.23	11.50	2.61	0.11	-0.48	-1.31
Zhwiki	4	4	1.33	7.84	1.52	1.23	11.90	2.61	0.14	-0.59	-1.59
Zhwiki	16	2	1.14	8.09	3.11	1.57	9.13	2.64	-0.64	-0.15	0.69
Zhwiki	16	3	.92	8.13	3.13	1.06	8.55	2.61	-0.20	-0.06	0.77
Zhwiki	16	4	1.13	6.37	1.98	1.08	8.88	2.62	0.06	-0.37	-0.93
Zhwiki	32	2	1.22	6.52	2.56	1.57	8.49	2.63	-0.51	-0.29	-0.09
Zhwiki	32	3	.92	7.42	2.95	1.04	7.93	2.61	-0.18	-0.07	0.50
Zhwiki	32	4	1.08	6.30	2.24	1.06	7.96	2.60	0.02	-0.24	-0.54
								μ	**-0.19**	-0.24	-0.36
								σ	0.33	0.22	0.79

Table D.135: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=URIOrEmail, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.24	79.50	24.00	2.40	75.10	16.10	-0.11	0.31	5.53
Arwiki	4	3	2.14	71.40	16.60	2.01	74.50	16.20	0.09	-0.22	0.29
Arwiki	4	4	2.08	67.90	15.30	2.02	74.80	16.20	0.05	-0.49	-0.66
Arwiki	16	2	2.01	52.60	23.40	2.24	55.30	16.10	-0.16	-0.19	5.10
Arwiki	16	3	1.62	52.20	21.90	1.70	56.70	16.20	-0.06	-0.31	3.97
Arwiki	16	4	1.87	42.20	14.10	1.71	56.30	16.20	0.11	-0.99	-1.47
Arwiki	32	2	1.85	52.80	27.50	2.25	53.50	16.10	-0.28	-0.05	7.98
Arwiki	32	3	1.64	42.80	19.30	1.66	53.10	16.20	-0.02	-0.72	2.20
Arwiki	32	4	1.79	37.70	14.30	1.66	53.30	16.20	0.09	-1.09	-1.27
Enwiki	4	2	.77	43.40	11.00	1.28	26.60	.01	-0.51	1.69	11.10
Enwiki	4	3	.66	39.70	7.23	.89	28.10	.01	-0.23	1.18	7.27
Enwiki	4	4	.62	32.50	5.41	.91	27.10	.01	-0.29	0.54	5.44
Enwiki	16	2	.67	26.10	11.10	1.19	11.20	.01	-0.53	1.50	11.20
Enwiki	16	3	.52	22.90	7.23	.71	12.70	.01	-0.19	1.03	7.28
Enwiki	16	4	.52	20.20	5.45	.74	12.50	.01	-0.23	0.77	5.48
Enwiki	32	2	.63	18.30	11.10	1.20	10.10	.01	-0.57	0.82	11.20
Enwiki	32	3	.49	18.40	7.24	.69	10.30	.01	-0.20	0.82	7.28
Enwiki	32	4	.49	15.50	5.45	.70	10.30	.01	-0.21	0.52	5.48
Ruwiki	4	2	2.57	99.60	30.70	2.92	87.90	17.80	-0.20	0.66	7.30
Ruwiki	4	3	2.46	90.50	22.00	2.42	87.10	17.90	0.02	0.19	2.35
Ruwiki	4	4	2.54	84.10	17.80	2.42	87.80	17.90	0.07	-0.21	-0.05
Ruwiki	16	2	2.23	71.00	32.90	2.73	63.80	17.80	-0.28	0.41	8.55
Ruwiki	16	3	2.07	54.90	20.90	2.03	65.60	17.90	0.02	-0.60	1.69
Ruwiki	16	4	2.24	55.70	17.80	2.05	65.10	17.90	0.11	-0.53	-0.02
Ruwiki	32	2	2.30	58.70	29.10	2.73	62.00	17.80	-0.24	-0.19	6.42
Ruwiki	32	3	1.98	48.40	20.90	1.97	61.20	17.80	0.01	-0.73	1.78
Ruwiki	32	4	2.15	50.20	17.70	1.98	61.80	17.90	0.10	-0.66	-0.13
Zhwiki	4	2	1.00	44.70	15.40	1.20	37.30	8.23	-0.29	1.10	10.50
Zhwiki	4	3	1.07	35.60	8.65	1.00	36.90	8.29	0.10	-0.19	0.53
Zhwiki	4	4	1.10	33.80	7.24	1.01	37.40	8.29	0.12	-0.53	-1.54
Zhwiki	16	2	.89	31.70	15.00	1.13	28.10	8.24	-0.34	0.53	9.97
Zhwiki	16	3	.76	29.40	13.10	.85	28.60	8.30	-0.15	0.12	6.99
Zhwiki	16	4	.92	24.20	8.63	.87	28.50	8.31	0.07	-0.63	0.47
Zhwiki	32	2	.95	26.00	13.10	1.12	27.10	8.28	-0.26	-0.16	7.07
Zhwiki	32	3	.75	25.70	12.50	.83	27.00	8.30	-0.12	-0.19	6.16
Zhwiki	32	4	.86	23.20	9.30	.85	27.30	8.30	0.01	-0.60	1.47
								μ	**-0.12**	0.08	4.52
								σ	0.19	0.71	3.94

Table D.136: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=URIOrEmail, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.58	40.30	17.80	2.27	43.90	13.30	0.22	-0.25	3.19
Arwiki	4	3	2.70	35.30	12.60	2.19	44.00	13.30	0.36	-0.60	-0.45
Arwiki	4	4	2.58	31.60	11.70	2.19	43.30	13.30	0.27	-0.82	-1.06
Arwiki	16	2	2.19	27.00	17.40	1.84	33.10	13.30	0.24	-0.43	2.91
Arwiki	16	3	1.83	25.70	16.90	1.67	33.30	13.30	0.11	-0.53	2.53
Arwiki	16	4	2.18	18.40	10.70	1.70	32.60	13.20	0.33	-1.00	-1.76
Arwiki	32	2	1.87	25.40	20.70	1.80	30.40	13.30	0.05	-0.35	5.16
Arwiki	32	3	1.89	19.90	14.70	1.58	30.60	13.30	0.21	-0.75	1.03
Arwiki	32	4	2.07	15.30	10.90	1.61	30.40	13.20	0.32	-1.05	-1.64
Enwiki	4	2	.71	21.40	7.79	1.00	13.80	.00	-0.29	0.77	7.85
Enwiki	4	3	.60	18.00	5.19	.88	14.90	.00	-0.28	0.31	5.23
Enwiki	4	4	.57	13.50	3.89	.90	13.80	.00	-0.34	-0.03	3.91
Enwiki	16	2	.53	13.20	7.80	.86	6.78	.00	-0.34	0.65	7.86
Enwiki	16	3	.43	9.56	5.19	.60	6.61	.00	-0.17	0.30	5.23
Enwiki	16	4	.44	6.68	3.91	.63	6.64	.00	-0.19	0.00	3.94
Enwiki	32	2	.48	9.84	7.80	.85	4.60	.00	-0.38	0.53	7.86
Enwiki	32	3	.42	7.80	5.19	.57	4.69	.00	-0.15	0.31	5.23
Enwiki	32	4	.43	5.16	3.90	.59	4.68	.00	-0.16	0.05	3.93
Ruwiki	4	2	2.85	50.20	23.10	2.71	50.90	14.70	0.08	-0.04	4.80
Ruwiki	4	3	3.01	44.50	16.80	2.58	51.60	14.70	0.24	-0.40	1.23
Ruwiki	4	4	3.24	37.90	13.60	2.59	50.60	14.70	0.36	-0.72	-0.58
Ruwiki	16	2	2.22	36.30	24.70	2.19	37.90	14.70	0.02	-0.09	5.69
Ruwiki	16	3	2.43	26.40	15.80	1.95	38.20	14.70	0.27	-0.67	0.63
Ruwiki	16	4	2.46	23.30	13.60	2.00	37.50	14.70	0.27	-0.81	-0.58
Ruwiki	32	2	2.37	27.70	21.70	2.14	34.40	14.60	0.13	-0.38	3.97
Ruwiki	32	3	2.29	21.90	15.70	1.83	34.70	14.70	0.26	-0.72	0.62
Ruwiki	32	4	2.34	19.20	13.50	1.87	34.60	14.70	0.27	-0.87	-0.67
Zhwiki	4	2	1.07	23.20	11.80	1.12	21.40	6.78	-0.07	0.27	7.41
Zhwiki	4	3	1.33	17.50	6.61	1.07	21.60	6.78	0.37	-0.60	-0.26
Zhwiki	4	4	1.34	15.10	5.62	1.08	21.30	6.78	0.38	-0.91	-1.71
Zhwiki	16	2	.87	16.40	11.50	.92	16.40	6.77	-0.07	0.00	6.90
Zhwiki	16	3	.79	14.60	10.10	.83	16.60	6.78	-0.06	-0.29	4.84
Zhwiki	16	4	.99	10.60	6.66	.84	16.40	6.77	0.22	-0.85	-0.17
Zhwiki	32	2	.95	12.70	9.79	.89	15.30	6.77	0.09	-0.38	4.43
Zhwiki	32	3	.78	12.30	9.59	.79	15.30	6.78	-0.00	-0.44	4.13
Zhwiki	32	4	.91	9.66	7.10	.80	15.30	6.78	0.16	-0.82	0.47
								μ	**0.08**	-0.32	2.84
								σ	0.23	0.48	3.00

Table D.137: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=URIOrEmail, Colours=never

Arguments: Xquote -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	3.37	17.20	3.64	4.28	22.50	4.96	-0.63	-0.37	-0.92
Arwiki	4	3	2.98	21.80	5.19	2.98	21.10	4.91	-0.00	0.05	0.20
Arwiki	4	4	3.30	17.60	3.02	3.02	22.00	4.85	0.20	-0.31	-1.28
Arwiki	16	2	2.78	16.10	5.91	4.07	16.70	4.98	-0.90	-0.04	0.65
Arwiki	16	3	2.60	16.40	5.45	2.64	16.00	4.94	-0.02	0.03	0.36
Arwiki	16	4	3.15	12.80	3.35	2.67	16.30	4.89	0.34	-0.24	-1.07
Arwiki	32	2	3.02	12.30	4.41	4.09	15.30	5.00	-0.75	-0.21	-0.41
Arwiki	32	3	2.57	15.30	5.28	2.61	15.10	4.95	-0.02	0.02	0.23
Arwiki	32	4	3.18	10.40	2.82	2.63	15.10	4.91	0.38	-0.33	-1.46
Enwiki	4	2	1.59	7.49	.63	2.71	5.98	.05	-1.13	0.15	0.58
Enwiki	4	3	1.58	7.47	.60	1.65	5.04	.05	-0.07	0.24	0.55
Enwiki	4	4	1.63	6.39	.31	1.68	5.92	.05	-0.05	0.05	0.26
Enwiki	16	2	1.46	4.27	.65	2.57	2.10	.05	-1.12	0.22	0.61
Enwiki	16	3	1.45	4.41	.73	1.49	1.49	.05	-0.04	0.29	0.68
Enwiki	16	4	1.50	3.49	.30	1.49	1.78	.05	0.01	0.17	0.26
Enwiki	32	2	1.43	3.79	.66	2.58	.97	.05	-1.17	0.28	0.61
Enwiki	32	3	1.45	3.92	.76	1.45	.86	.05	0.01	0.31	0.71
Enwiki	32	4	1.49	2.94	.34	1.46	.97	.05	0.03	0.20	0.30
Ruwiki	4	2	3.69	23.80	6.10	5.40	25.00	5.21	-0.97	-0.07	0.51
Ruwiki	4	3	3.99	20.30	3.37	3.63	23.10	5.16	0.20	-0.16	-1.01
Ruwiki	4	4	3.89	23.30	4.29	3.73	24.30	5.11	0.09	-0.06	-0.47
Ruwiki	16	2	3.55	15.70	5.42	5.14	17.90	5.22	-0.90	-0.12	0.11
Ruwiki	16	3	3.44	15.80	4.48	3.21	16.80	5.16	0.13	-0.06	-0.38
Ruwiki	16	4	3.92	13.80	2.96	3.25	17.20	5.13	0.38	-0.19	-1.23
Ruwiki	32	2	3.73	12.50	4.13	5.15	16.00	5.23	-0.81	-0.20	-0.62
Ruwiki	32	3	3.40	12.90	3.53	3.19	15.60	5.13	0.12	-0.15	-0.90
Ruwiki	32	4	3.83	12.60	3.13	3.21	15.80	5.13	0.35	-0.18	-1.13
Zhwiki	4	2	1.73	8.15	1.71	2.27	10.30	2.37	-0.79	-0.32	-0.98
Zhwiki	4	3	1.56	8.88	1.89	1.53	9.62	2.34	0.04	-0.11	-0.66
Zhwiki	4	4	1.63	7.83	1.40	1.55	10.10	2.34	0.12	-0.34	-1.37
Zhwiki	16	2	1.49	7.06	2.58	2.16	7.59	2.38	-0.99	-0.08	0.29
Zhwiki	16	3	1.48	5.31	1.56	1.37	7.21	2.35	0.16	-0.28	-1.16
Zhwiki	16	4	1.38	7.06	2.23	1.39	7.37	2.34	-0.01	-0.05	-0.16
Zhwiki	32	2	1.39	7.86	3.11	2.16	6.93	2.38	-1.12	0.14	1.07
Zhwiki	32	3	1.46	4.76	1.44	1.35	6.77	2.34	0.15	-0.30	-1.32
Zhwiki	32	4	1.53	5.01	1.48	1.37	6.80	2.33	0.23	-0.26	-1.24
								μ	**-0.24**	-0.06	-0.27
								σ	0.52	0.20	0.77

Table D.138: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=Xquote, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.54	84.10	23.10	2.92	76.10	15.60	-0.27	0.56	5.22
Arwiki	4	3	2.18	87.30	24.10	2.31	75.30	15.70	-0.08	0.84	5.83
Arwiki	4	4	2.37	71.40	15.10	2.32	76.60	15.70	0.03	-0.36	-0.43
Arwiki	16	2	2.08	66.30	30.50	2.77	56.80	15.70	-0.48	0.67	10.30
Arwiki	16	3	1.86	60.50	24.10	1.98	57.80	15.70	-0.08	0.19	5.90
Arwiki	16	4	2.11	46.00	16.10	1.99	57.50	15.70	0.08	-0.81	0.27
Arwiki	32	2	2.22	53.90	25.10	2.77	54.70	15.70	-0.39	-0.05	6.54
Arwiki	32	3	1.83	54.10	23.90	1.94	54.40	15.70	-0.07	-0.02	5.69
Arwiki	32	4	2.12	38.00	13.90	1.95	54.90	15.70	0.12	-1.18	-1.29
Enwiki	4	2	1.03	45.80	11.10	1.70	26.00	.08	-0.68	1.99	11.10
Enwiki	4	3	.93	41.80	7.28	1.11	25.90	.10	-0.18	1.61	7.24
Enwiki	4	4	.90	36.60	5.51	1.13	26.30	.09	-0.23	1.04	5.45
Enwiki	16	2	.90	29.20	11.20	1.60	11.10	.08	-0.70	1.83	11.20
Enwiki	16	3	.80	26.80	7.27	.93	12.50	.08	-0.13	1.44	7.25
Enwiki	16	4	.80	21.40	5.51	.94	12.40	.09	-0.15	0.91	5.46
Enwiki	32	2	.86	26.60	11.10	1.60	10.30	.08	-0.74	1.65	11.20
Enwiki	32	3	.79	22.90	7.29	.91	10.40	.08	-0.12	1.25	7.26
Enwiki	32	4	.79	18.50	5.52	.93	10.70	.09	-0.14	0.79	5.47
Ruwiki	4	2	2.76	108.00	34.40	3.67	83.70	16.40	-0.52	1.39	10.20
Ruwiki	4	3	2.83	89.80	19.60	2.76	83.10	16.50	0.04	0.38	1.75
Ruwiki	4	4	2.70	92.00	21.20	2.78	84.40	16.50	-0.05	0.43	2.70
Ruwiki	16	2	2.57	72.60	31.50	3.45	59.50	16.40	-0.50	0.74	8.59
Ruwiki	16	3	2.34	62.80	22.60	2.37	61.20	16.50	-0.02	0.09	3.47
Ruwiki	16	4	2.57	53.40	16.20	2.38	61.00	16.50	0.11	-0.43	-0.17
Ruwiki	32	2	2.68	60.40	26.80	3.46	57.30	16.40	-0.44	0.17	5.91
Ruwiki	32	3	2.38	51.60	19.40	2.31	57.40	16.40	0.04	-0.33	1.69
Ruwiki	32	4	2.50	47.00	16.20	2.32	57.60	16.50	0.10	-0.60	-0.16
Zhwiki	4	2	1.29	38.50	10.80	1.53	35.60	7.50	-0.36	0.42	4.79
Zhwiki	4	3	1.17	38.20	9.20	1.16	35.20	7.51	0.00	0.44	2.48
Zhwiki	4	4	1.20	34.00	6.95	1.18	35.80	7.54	0.02	-0.26	-0.85
Zhwiki	16	2	1.09	30.50	13.40	1.45	26.40	7.50	-0.53	0.60	8.71
Zhwiki	16	3	1.06	23.20	7.93	1.01	26.90	7.51	0.07	-0.54	0.61
Zhwiki	16	4	.99	25.80	9.58	1.03	26.90	7.53	-0.05	-0.17	3.00
Zhwiki	32	2	1.02	31.20	14.90	1.45	25.50	7.50	-0.63	0.83	10.90
Zhwiki	32	3	1.05	19.80	7.66	.99	25.40	7.51	0.09	-0.82	0.23
Zhwiki	32	4	1.05	19.20	6.98	1.00	25.70	7.53	0.07	-0.95	-0.80
								μ	**-0.19**	0.38	4.79
								σ	0.26	0.82	3.88

Table D.139: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=Xquote, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	2.77	42.40	17.30	2.54	45.00	12.90	0.16	-0.18	3.09
Arwiki	4	3	2.36	44.20	18.60	2.40	45.00	12.90	-0.03	-0.06	4.01
Arwiki	4	4	2.68	33.40	11.60	2.41	44.50	12.90	0.19	-0.77	-0.92
Arwiki	16	2	1.88	33.40	23.10	2.12	33.70	12.90	-0.17	-0.02	7.11
Arwiki	16	3	1.75	29.00	18.70	1.85	33.70	12.90	-0.07	-0.33	4.04
Arwiki	16	4	2.09	21.20	12.40	1.88	33.40	12.90	0.15	-0.85	-0.35
Arwiki	32	2	2.18	24.10	18.60	2.10	31.00	12.90	0.05	-0.48	4.00
Arwiki	32	3	1.68	24.80	18.40	1.76	30.90	12.90	-0.06	-0.42	3.84
Arwiki	32	4	2.17	15.90	10.50	1.79	31.00	12.90	0.26	-1.06	-1.66
Enwiki	4	2	.89	23.40	7.87	1.30	15.10	.07	-0.41	0.83	7.86
Enwiki	4	3	.77	19.80	5.24	1.05	15.20	.07	-0.29	0.46	5.21
Enwiki	4	4	.71	16.00	3.96	1.06	14.70	.08	-0.36	0.14	3.91
Enwiki	16	2	.67	14.50	7.87	1.12	7.12	.07	-0.45	0.74	7.86
Enwiki	16	3	.51	11.10	5.24	.74	6.78	.07	-0.23	0.43	5.21
Enwiki	16	4	.45	8.06	3.98	.76	6.96	.07	-0.32	0.11	3.93
Enwiki	32	2	.63	10.60	7.86	1.11	4.98	.06	-0.48	0.57	7.86
Enwiki	32	3	.44	9.03	5.24	.68	5.01	.06	-0.24	0.41	5.22
Enwiki	32	4	.44	6.30	3.97	.76	5.03	.07	-0.32	0.13	3.93
Ruwiki	4	2	2.72	55.30	26.00	3.07	49.50	13.40	-0.20	0.33	7.14
Ruwiki	4	3	3.21	44.80	14.70	2.80	50.00	13.40	0.23	-0.29	0.73
Ruwiki	4	4	2.82	42.10	16.40	2.81	49.30	13.40	0.00	-0.41	1.65
Ruwiki	16	2	2.41	35.90	23.60	2.60	35.60	13.50	-0.11	0.02	5.73
Ruwiki	16	3	2.30	29.50	17.00	2.14	35.80	13.40	0.09	-0.36	2.02
Ruwiki	16	4	2.51	22.90	12.20	2.19	35.50	13.40	0.18	-0.71	-0.69
Ruwiki	32	2	2.63	25.90	19.60	2.58	32.20	13.40	0.03	-0.36	3.50
Ruwiki	32	3	2.37	22.10	14.50	2.03	32.20	13.40	0.20	-0.57	0.63
Ruwiki	32	4	2.39	19.00	12.20	2.07	32.20	13.40	0.18	-0.75	-0.69
Zhwiki	4	2	1.40	20.20	8.04	1.28	21.00	6.15	0.18	-0.12	2.77
Zhwiki	4	3	1.26	18.70	7.08	1.17	21.10	6.16	0.13	-0.34	1.35
Zhwiki	4	4	1.31	15.60	5.36	1.18	20.80	6.16	0.19	-0.77	-1.17
Zhwiki	16	2	1.01	15.50	10.10	1.09	15.50	6.14	-0.12	-0.00	5.82
Zhwiki	16	3	1.11	10.80	5.95	.92	15.60	6.15	0.28	-0.70	-0.29
Zhwiki	16	4	.88	11.70	7.43	.93	15.50	6.15	-0.07	-0.55	1.88
Zhwiki	32	2	.86	14.60	11.30	1.08	14.30	6.14	-0.31	0.05	7.51
Zhwiki	32	3	1.08	8.66	5.74	.87	14.30	6.15	0.32	-0.83	-0.60
Zhwiki	32	4	1.03	8.04	5.30	.89	14.30	6.15	0.21	-0.92	-1.24
								μ	**-0.03**	-0.21	3.06
								σ	0.23	0.49	2.95

Table D.140: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=Xquote, Colours=never

Arguments: \p{Greek} -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	1.46	10.00	.85	2.34	8.67	.04	-0.61	0.10	0.57
Arwiki	4	3	1.18	9.03	.98	1.58	7.11	.04	-0.28	0.13	0.66
Arwiki	4	4	1.32	10.90	.43	1.68	8.99	.04	-0.25	0.13	0.28
Arwiki	16	2	1.29	4.44	.90	2.16	2.83	.03	-0.60	0.11	0.61
Arwiki	16	3	1.06	4.31	.92	1.37	1.88	.04	-0.21	0.17	0.62
Arwiki	16	4	1.04	3.49	.39	1.39	2.70	.04	-0.24	0.06	0.25
Arwiki	32	2	1.30	3.01	.91	2.15	1.10	.03	-0.59	0.13	0.61
Arwiki	32	3	.98	3.30	.97	1.35	.95	.04	-0.26	0.16	0.65
Arwiki	32	4	1.04	2.31	.43	1.34	1.12	.03	-0.21	0.08	0.28
Enwiki	4	2	.89	6.85	.56	1.42	5.92	.02	-0.54	0.09	0.54
Enwiki	4	3	.79	5.48	.68	.99	5.07	.02	-0.20	0.04	0.67
Enwiki	4	4	.80	4.57	.30	1.02	6.22	.02	-0.22	-0.17	0.29
Enwiki	16	2	.75	2.96	.60	1.28	1.97	.02	-0.54	0.10	0.59
Enwiki	16	3	.71	2.60	.68	.81	1.29	.02	-0.10	0.13	0.67
Enwiki	16	4	.71	1.66	.27	.82	1.73	.02	-0.11	-0.01	0.25
Enwiki	32	2	.75	2.08	.62	1.29	.78	.01	-0.55	0.13	0.61
Enwiki	32	3	.71	2.15	.72	.79	.65	.02	-0.09	0.15	0.71
Enwiki	32	4	.69	1.27	.31	.80	.79	.02	-0.10	0.05	0.29
Ruwiki	4	2	1.92	12.80	1.28	3.11	11.10	.19	-0.68	0.10	0.62
Ruwiki	4	3	1.67	12.70	1.29	2.09	9.08	.19	-0.24	0.21	0.62
Ruwiki	4	4	1.95	17.10	.65	2.18	11.40	.19	-0.13	0.32	0.26
Ruwiki	16	2	1.72	5.98	1.32	2.89	3.92	.19	-0.66	0.12	0.64
Ruwiki	16	3	1.38	5.97	1.28	1.80	2.71	.19	-0.24	0.18	0.62
Ruwiki	16	4	1.37	6.73	.66	1.85	3.71	.19	-0.27	0.17	0.27
Ruwiki	32	2	1.70	4.55	1.44	2.86	1.83	.19	-0.66	0.15	0.71
Ruwiki	32	3	1.21	4.57	1.39	1.81	1.60	.19	-0.34	0.17	0.68
Ruwiki	32	4	1.33	4.34	.70	1.79	1.82	.18	-0.26	0.14	0.29
Zhwiki	4	2	.86	5.12	.44	1.44	4.19	.05	-0.84	0.14	0.57
Zhwiki	4	3	.77	6.08	.47	.92	3.45	.05	-0.22	0.39	0.61
Zhwiki	4	4	.91	7.94	.24	.96	4.28	.05	-0.08	0.54	0.27
Zhwiki	16	2	.79	2.92	.48	1.34	1.47	.05	-0.80	0.21	0.62
Zhwiki	16	3	.65	3.44	.50	.81	.99	.05	-0.24	0.36	0.66
Zhwiki	16	4	.82	5.87	.24	.83	1.39	.05	-0.01	0.66	0.28
Zhwiki	32	2	.80	2.53	.51	1.33	.66	.05	-0.78	0.28	0.67
Zhwiki	32	3	.57	2.69	.55	.79	.57	.05	-0.32	0.31	0.73
Zhwiki	32	4	.80	5.53	.27	.81	.66	.05	-0.02	0.72	0.31
								μ	**-0.35**	0.19	0.52
								σ	0.24	0.17	0.17

Table D.141: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=\p{Greek}, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.97	59.40	15.80	1.47	38.10	.04	-0.35	1.49	11.00
Arwiki	4	3	.77	47.50	10.50	1.12	38.90	.05	-0.24	0.61	7.26
Arwiki	4	4	.76	38.40	7.86	1.19	36.80	.05	-0.30	0.11	5.45
Arwiki	16	2	.78	36.00	16.00	1.34	15.30	.04	-0.39	1.45	11.10
Arwiki	16	3	.65	26.80	10.50	.86	17.70	.05	-0.15	0.64	7.27
Arwiki	16	4	.65	18.30	7.88	.92	17.10	.05	-0.19	0.08	5.46
Arwiki	32	2	.77	26.70	16.00	1.34	14.10	.04	-0.40	0.88	11.10
Arwiki	32	3	.64	22.40	10.50	.83	14.60	.05	-0.14	0.54	7.28
Arwiki	32	4	.64	16.30	7.88	.85	14.80	.05	-0.15	0.10	5.47
Enwiki	4	2	.63	39.30	11.00	1.00	26.30	.01	-0.37	1.31	11.00
Enwiki	4	3	.54	32.10	7.24	.76	26.00	.02	-0.23	0.62	7.28
Enwiki	4	4	.53	25.50	5.44	.80	25.10	.02	-0.28	0.04	5.47
Enwiki	16	2	.52	23.90	11.10	.90	10.30	.01	-0.38	1.37	11.20
Enwiki	16	3	.45	17.70	7.25	.57	12.00	.01	-0.12	0.57	7.29
Enwiki	16	4	.45	12.30	5.44	.61	11.50	.02	-0.15	0.08	5.46
Enwiki	32	2	.49	18.80	11.10	.90	9.67	.02	-0.41	0.92	11.20
Enwiki	32	3	.44	15.50	7.26	.55	9.91	.02	-0.11	0.56	7.30
Enwiki	32	4	.45	10.40	5.44	.57	10.10	.02	-0.12	0.03	5.46
Ruwiki	4	2	1.33	74.60	19.90	2.05	47.00	.45	-0.41	1.56	11.00
Ruwiki	4	3	1.11	66.40	13.10	1.50	47.00	.47	-0.22	1.10	7.16
Ruwiki	4	4	1.17	61.00	9.92	1.58	45.60	.48	-0.23	0.87	5.34
Ruwiki	16	2	1.11	45.10	20.00	1.85	19.80	.44	-0.42	1.43	11.10
Ruwiki	16	3	.85	34.20	13.10	1.18	22.40	.46	-0.19	0.67	7.19
Ruwiki	16	4	.87	28.10	9.94	1.24	21.90	.47	-0.21	0.35	5.36
Ruwiki	32	2	1.05	35.40	20.10	1.85	18.60	.45	-0.45	0.95	11.10
Ruwiki	32	3	.81	29.50	13.10	1.14	19.10	.46	-0.19	0.59	7.18
Ruwiki	32	4	.83	22.20	9.95	1.16	19.50	.47	-0.19	0.16	5.37
Zhwiki	4	2	.57	29.70	7.61	.90	18.00	.09	-0.48	1.72	11.00
Zhwiki	4	3	.51	28.50	5.04	.63	18.00	.10	-0.18	1.53	7.25
Zhwiki	4	4	.59	29.20	3.80	.66	17.40	.10	-0.11	1.73	5.43
Zhwiki	16	2	.48	18.30	7.64	.83	7.57	.09	-0.51	1.57	11.10
Zhwiki	16	3	.40	16.70	5.05	.51	8.54	.09	-0.16	1.19	7.28
Zhwiki	16	4	.49	20.90	3.81	.54	8.35	.10	-0.07	1.84	5.45
Zhwiki	32	2	.47	13.90	7.67	.82	7.03	.09	-0.52	1.01	11.10
Zhwiki	32	3	.34	13.10	5.06	.49	7.19	.09	-0.23	0.87	7.29
Zhwiki	32	4	.48	19.00	3.81	.51	7.33	.10	-0.05	1.71	5.44
									μ	**-0.26**	0.90	7.93
									σ	0.13	0.56	2.36

Table D.142: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=\p{Greek}, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	.97	30.70	11.30	1.38	21.40	.03	-0.29	0.65	7.84
Arwiki	4	3	.92	26.10	7.52	1.29	22.60	.03	-0.26	0.25	5.22
Arwiki	4	4	.87	19.80	5.65	1.34	21.50	.04	-0.33	-0.12	3.92
Arwiki	16	2	.69	18.90	11.20	1.14	10.00	.03	-0.32	0.62	7.82
Arwiki	16	3	.62	13.90	7.51	.85	9.81	.04	-0.16	0.29	5.22
Arwiki	16	4	.63	8.96	5.63	.88	9.84	.04	-0.18	-0.06	3.90
Arwiki	32	2	.60	13.10	11.20	1.13	6.85	.03	-0.37	0.44	7.82
Arwiki	32	3	.61	11.40	7.51	.78	7.01	.04	-0.12	0.30	5.22
Arwiki	32	4	.62	6.93	5.63	.83	6.98	.04	-0.14	-0.00	3.91
Enwiki	4	2	.68	21.20	7.78	.97	14.70	.01	-0.29	0.65	7.84
Enwiki	4	3	.61	17.80	5.19	.90	15.40	.01	-0.29	0.23	5.22
Enwiki	4	4	.57	13.70	3.89	.93	14.60	.01	-0.35	-0.10	3.91
Enwiki	16	2	.46	13.00	7.77	.79	6.87	.01	-0.34	0.61	7.83
Enwiki	16	3	.43	9.35	5.19	.59	6.71	.01	-0.16	0.27	5.22
Enwiki	16	4	.43	6.33	3.89	.60	6.66	.01	-0.18	-0.03	3.91
Enwiki	32	2	.42	9.38	7.77	.78	4.69	.01	-0.36	0.47	7.82
Enwiki	32	3	.41	7.66	5.18	.53	4.78	.01	-0.12	0.29	5.21
Enwiki	32	4	.42	4.95	3.89	.56	4.77	.01	-0.13	0.02	3.91
Ruwiki	4	2	1.33	38.90	14.20	1.91	27.20	.38	-0.33	0.66	7.83
Ruwiki	4	3	1.31	33.60	9.43	1.70	28.00	.39	-0.22	0.32	5.12
Ruwiki	4	4	1.25	26.40	7.12	1.75	27.10	.41	-0.29	-0.04	3.80
Ruwiki	16	2	.95	23.90	14.10	1.59	13.10	.37	-0.36	0.61	7.76
Ruwiki	16	3	.80	17.60	9.47	1.15	12.50	.39	-0.20	0.29	5.14
Ruwiki	16	4	.80	11.80	7.13	1.20	12.70	.40	-0.23	-0.06	3.81
Ruwiki	32	2	.87	16.90	14.10	1.57	9.18	.35	-0.40	0.44	7.81
Ruwiki	32	3	.77	14.40	9.46	1.07	9.36	.38	-0.17	0.29	5.14
Ruwiki	32	4	.79	9.50	7.13	1.12	9.33	.39	-0.19	0.01	3.82
Zhwiki	4	2	.54	15.20	5.41	.79	10.40	.10	-0.37	0.70	7.80
Zhwiki	4	3	.53	13.10	3.62	.67	10.70	.10	-0.21	0.36	5.16
Zhwiki	4	4	.52	10.70	2.73	.69	10.30	.09	-0.24	0.06	3.88
Zhwiki	16	2	.39	9.40	5.41	.66	4.95	.07	-0.41	0.65	7.84
Zhwiki	16	3	.32	6.95	3.64	.47	4.71	.08	-0.22	0.33	5.22
Zhwiki	16	4	.40	6.68	2.74	.48	4.79	.08	-0.13	0.28	3.90
Zhwiki	32	2	.35	6.72	5.40	.65	3.47	.07	-0.44	0.48	7.83
Zhwiki	32	3	.29	5.62	3.64	.43	3.52	.07	-0.21	0.31	5.23
Zhwiki	32	4	.37	5.12	2.73	.46	3.51	.08	-0.14	0.24	3.89
									μ	**-0.25**	0.30	5.63
									σ	0.09	0.25	1.64

Table D.143: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=\p{Greek}, Colours=never

Arguments: \X -colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	55.20	63.50	16.70	61.70	55.40	21.40	-4.50	0.57	-3.33
Arwiki	4	3	55.40	64.80	20.70	54.90	54.10	21.30	0.32	0.75	-0.41
Arwiki	4	4	56.70	46.80	13.70	54.90	54.00	21.10	1.31	-0.50	-5.17
Arwiki	16	2	55.20	44.90	14.70	60.60	49.80	21.50	-3.76	-0.34	-4.80
Arwiki	16	3	55.20	52.50	20.30	54.50	48.80	21.30	0.45	0.25	-0.67
Arwiki	16	4	56.70	34.80	11.40	54.60	48.70	21.10	1.43	-0.97	-6.74
Arwiki	32	2	54.60	45.80	16.00	60.50	48.70	21.60	-4.10	-0.20	-3.95
Arwiki	32	3	56.00	31.60	10.90	54.40	48.10	21.30	1.10	-1.15	-7.30
Arwiki	32	4	56.80	33.00	11.40	54.50	47.90	21.20	1.61	-1.04	-6.82
Enwiki	4	2	37.40	44.80	13.40	41.20	31.00	12.70	-3.79	1.38	0.66
Enwiki	4	3	39.40	27.60	7.16	37.90	30.50	12.60	1.51	-0.30	-5.45
Enwiki	4	4	38.60	37.30	12.60	38.00	30.50	12.50	0.58	0.68	0.14
Enwiki	16	2	36.90	38.70	14.50	40.30	27.50	12.70	-3.35	1.13	1.75
Enwiki	16	3	38.70	23.10	7.87	37.70	27.10	12.60	1.03	-0.40	-4.80
Enwiki	16	4	38.00	32.90	13.20	37.60	27.00	12.50	0.42	0.60	0.73
Enwiki	32	2	36.80	36.60	13.90	40.30	26.80	12.70	-3.52	0.98	1.17
Enwiki	32	3	38.40	21.40	7.39	37.60	26.50	12.60	0.81	-0.51	-5.23
Enwiki	32	4	38.00	31.40	12.90	37.70	26.40	12.50	0.26	0.51	0.44
Ruwiki	4	2	67.00	83.50	23.20	77.30	65.90	23.80	-5.82	1.00	-0.34
Ruwiki	4	3	68.60	65.90	18.60	67.80	64.40	23.60	0.45	0.08	-2.78
Ruwiki	4	4	69.00	63.60	18.20	67.70	64.40	23.30	0.73	-0.04	-2.93
Ruwiki	16	2	67.00	58.20	19.30	75.40	59.20	23.90	-4.74	-0.05	-2.62
Ruwiki	16	3	67.30	60.90	23.20	67.20	57.90	23.60	0.06	0.17	-0.22
Ruwiki	16	4	68.60	47.30	16.50	67.20	57.90	23.40	0.83	-0.60	-3.92
Ruwiki	32	2	66.70	56.00	19.10	75.10	57.70	23.90	-4.77	-0.10	-2.74
Ruwiki	32	3	68.10	46.50	16.70	67.00	57.00	23.70	0.65	-0.59	-3.99
Ruwiki	32	4	68.60	38.60	13.20	67.10	56.50	23.40	0.87	-1.01	-5.73
Zhwiki	4	2	26.30	31.90	8.06	31.90	27.40	10.60	-8.20	0.66	-3.77
Zhwiki	4	3	26.70	27.40	7.91	26.30	26.90	10.50	0.66	0.08	-3.87
Zhwiki	4	4	28.10	19.10	4.77	26.20	26.80	10.40	2.71	-1.13	-8.31
Zhwiki	16	2	25.70	28.30	10.30	31.10	24.90	10.70	-7.95	0.51	-0.59
Zhwiki	16	3	26.90	18.00	6.14	25.90	24.40	10.60	1.38	-0.94	-6.50
Zhwiki	16	4	27.20	22.50	8.40	26.00	24.30	10.40	1.75	-0.26	-3.00
Zhwiki	32	2	25.60	28.00	10.40	31.10	24.30	10.70	-8.07	0.54	-0.34
Zhwiki	32	3	26.50	19.70	7.34	25.90	24.10	10.60	0.89	-0.64	-4.76
Zhwiki	32	4	27.40	14.70	5.12	26.00	23.80	10.50	2.08	-1.35	-7.88
								μ	**-1.07**	-0.06	-3.17
								σ	3.14	0.72	2.77

Table D.144: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `SSE-4.2` with arguments Expression=\X, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	17.50	253.00	61.90	19.90	188.00	67.30	-1.72	4.52	-3.76
Arwiki	4	3	17.10	240.00	73.10	16.30	189.00	67.60	0.53	3.59	3.89
Arwiki	4	4	17.80	181.00	47.80	16.30	188.00	67.80	1.04	-0.50	-13.90
Arwiki	16	2	17.60	172.00	49.50	19.70	166.00	67.40	-1.48	0.38	-12.50
Arwiki	16	3	16.90	182.00	67.90	16.10	167.00	67.60	0.58	1.06	0.22
Arwiki	16	4	17.90	130.00	36.20	16.10	167.00	67.70	1.27	-2.57	-22.00
Arwiki	32	2	17.30	184.00	56.90	19.50	164.00	67.70	-1.58	1.43	-7.54
Arwiki	32	3	17.90	126.00	37.20	16.10	164.00	68.40	1.26	-2.67	-21.80
Arwiki	32	4	17.80	116.00	37.70	16.10	164.00	68.30	1.22	-3.32	-21.30
Enwiki	4	2	11.40	187.00	49.50	13.10	122.00	38.80	-1.74	6.56	10.70
Enwiki	4	3	12.10	127.00	26.00	11.00	123.00	38.90	1.08	0.35	-13.10
Enwiki	4	4	11.40	142.00	44.40	11.00	123.00	38.90	0.42	1.84	5.47
Enwiki	16	2	11.10	150.00	49.60	12.80	108.00	39.00	-1.71	4.20	10.60
Enwiki	16	3	11.90	96.50	25.90	10.90	109.00	39.20	1.01	-1.24	-13.40
Enwiki	16	4	11.20	115.00	44.50	10.90	110.00	39.20	0.39	0.57	5.40
Enwiki	32	2	11.10	152.00	49.70	12.80	106.00	39.20	-1.66	4.61	10.50
Enwiki	32	3	11.80	92.20	26.10	10.80	106.00	39.60	0.98	-1.41	-13.60
Enwiki	32	4	11.20	113.00	44.50	10.80	107.00	39.60	0.39	0.60	4.92
Ruwiki	4	2	20.70	332.00	86.30	25.50	212.00	73.70	-2.72	6.79	7.17
Ruwiki	4	3	21.20	261.00	66.50	19.80	212.00	73.70	0.77	2.74	-4.07
Ruwiki	4	4	21.30	233.00	63.20	19.90	212.00	74.00	0.79	1.21	-6.11
Ruwiki	16	2	20.90	222.00	65.60	25.00	186.00	73.90	-2.27	2.03	-4.68
Ruwiki	16	3	20.50	217.00	77.20	19.50	187.00	74.00	0.54	1.72	1.83
Ruwiki	16	4	21.20	167.00	53.50	19.50	187.00	74.00	0.93	-1.16	-11.60
Ruwiki	32	2	20.80	226.00	68.70	24.80	182.00	74.10	-2.28	2.51	-3.07
Ruwiki	32	3	21.10	172.00	57.50	19.50	182.00	74.60	0.86	-0.55	-9.67
Ruwiki	32	4	21.50	146.00	43.90	19.50	182.00	74.70	1.09	-2.06	-17.40
Zhwiki	4	2	8.37	137.00	30.50	11.00	90.90	32.90	-3.87	6.71	-3.47
Zhwiki	4	3	8.39	111.00	28.20	7.82	90.60	33.10	0.83	2.96	-7.07
Zhwiki	4	4	8.74	90.10	16.60	7.82	91.10	33.00	1.35	-0.14	-24.20
Zhwiki	16	2	8.06	109.00	34.90	10.80	80.50	32.90	-3.97	4.17	2.88
Zhwiki	16	3	8.49	73.40	20.20	7.69	80.80	33.10	1.18	-1.08	-18.90
Zhwiki	16	4	8.22	85.30	28.10	7.68	80.90	33.20	0.80	0.64	-7.44
Zhwiki	32	2	7.96	111.00	37.20	10.70	79.40	33.10	-4.06	4.68	6.06
Zhwiki	32	3	8.31	76.70	25.50	7.68	79.10	33.40	0.93	-0.35	-11.60
Zhwiki	32	4	8.56	58.80	17.00	7.68	78.90	33.40	1.30	-2.96	-24.20
								μ	**-0.21**	1.27	-6.30
								σ	1.68	2.74	10.40

Table D.145: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX2` with arguments Expression=\X, Colours=never

CONFIG			CURR			HYBRID			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	2	15.30	95.30	49.60	15.60	107.00	55.60	-0.22	-0.84	-4.25
Arwiki	4	3	14.50	104.00	59.20	13.30	107.00	55.60	0.82	-0.23	2.46
Arwiki	4	4	16.00	76.60	38.60	13.30	107.00	55.60	1.92	-2.09	-11.90
Arwiki	16	2	15.30	59.10	39.50	15.20	93.70	55.60	0.09	-2.42	-11.30
Arwiki	16	3	14.10	73.40	55.00	12.70	94.90	55.60	0.98	-1.50	-0.43
Arwiki	16	4	16.10	44.60	29.10	12.60	94.80	55.60	2.41	-3.51	-18.50
Arwiki	32	2	14.80	60.10	45.40	15.10	92.50	55.70	-0.20	-2.26	-7.15
Arwiki	32	3	16.10	41.10	29.90	12.60	91.90	55.60	2.44	-3.54	-18.00
Arwiki	32	4	16.00	41.10	30.30	12.50	92.40	55.60	2.42	-3.58	-17.70
Enwiki	4	2	9.27	72.40	39.70	10.40	70.20	32.00	-1.09	0.22	7.75
Enwiki	4	3	10.70	51.00	20.70	8.82	71.30	32.00	1.88	-2.05	-11.40
Enwiki	4	4	9.48	61.20	35.60	8.84	71.10	32.00	0.64	-0.99	3.63
Enwiki	16	2	8.76	54.50	39.60	10.00	61.20	32.00	-1.26	-0.67	7.71
Enwiki	16	3	10.20	33.90	20.70	8.40	62.20	32.00	1.85	-2.85	-11.40
Enwiki	16	4	9.03	45.80	35.60	8.36	61.20	32.00	0.67	-1.55	3.65
Enwiki	32	2	8.75	50.80	39.60	9.93	59.50	32.00	-1.19	-0.87	7.67
Enwiki	32	3	10.20	29.20	20.70	8.32	59.40	32.00	1.92	-3.04	-11.40
Enwiki	32	4	9.01	43.20	35.60	8.28	59.30	32.00	0.73	-1.63	3.64
Ruwiki	4	2	17.20	127.00	69.30	20.00	124.00	60.90	-1.62	0.16	4.77
Ruwiki	4	3	18.40	110.00	53.80	16.00	123.00	60.90	1.36	-0.74	-4.03
Ruwiki	4	4	18.50	99.00	51.30	16.00	123.00	60.90	1.43	-1.33	-5.39
Ruwiki	16	2	17.70	76.70	52.40	19.50	108.00	60.90	-0.98	-1.77	-4.78
Ruwiki	16	3	16.90	85.40	62.50	15.30	108.00	60.90	0.89	-1.26	0.90
Ruwiki	16	4	18.40	62.10	43.20	15.30	108.00	60.90	1.77	-2.62	-9.99
Ruwiki	32	2	17.50	73.00	54.90	19.30	104.00	60.90	-1.04	-1.78	-3.38
Ruwiki	32	3	18.10	60.50	46.20	15.10	103.00	60.90	1.70	-2.43	-8.30
Ruwiki	32	4	19.00	48.90	35.30	15.20	104.00	60.90	2.14	-3.12	-14.50
Zhwiki	4	2	7.39	47.80	24.50	8.51	53.10	27.10	-1.66	-0.78	-3.80
Zhwiki	4	3	7.37	44.70	22.70	6.52	51.80	27.10	1.24	-1.04	-6.39
Zhwiki	4	4	8.09	32.40	13.30	6.53	51.90	27.10	2.30	-2.86	-20.20
Zhwiki	16	2	6.70	37.70	28.00	8.24	45.30	27.10	-2.26	-1.12	1.35
Zhwiki	16	3	7.56	24.90	16.20	6.14	46.30	27.10	2.08	-3.14	-15.90
Zhwiki	16	4	7.03	30.60	22.70	6.12	45.80	27.10	1.34	-2.23	-6.37
Zhwiki	32	2	6.52	37.00	29.80	8.18	43.90	27.10	-2.44	-1.01	3.98
Zhwiki	32	3	7.21	25.90	20.40	6.04	44.40	27.10	1.72	-2.72	-9.85
Zhwiki	32	4	7.72	19.60	13.60	6.03	44.80	27.10	2.48	-3.70	-19.80
								μ	**0.70**	-1.86	-5.79
								σ	1.47	1.07	8.28

Table D.146: `icgrep` PAPI comparison between `CURR` and `HYBRID` on `AVX-512` with arguments Expression=\X, Colours=never

D.6 Comparison against non-Parabix applications

D.6.1 Case study: `base64` vs. `fastbase64`

CONFIG			FB64			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	.79	4.53	.01	2.21	9.03	3.78	-0.99	-0.31	-2.63
Enwiki	4	1	.55	3.08	.01	1.53	6.23	2.66	-0.99	-0.32	-2.67
Ruwiki	4	1	.98	5.60	.01	2.73	11.20	4.76	-0.99	-0.32	-2.69
Zhwiki	4	1	.38	2.12	.00	1.05	4.25	1.80	-0.99	-0.31	-2.63
								μ	**-0.99**	-0.32	-2.66
								σ	0.00	0.00	0.03

Table D.147: `base64` PAPI comparison between `FB64` and `CURR` on `SSE-4.2`

CONFIG			FB64			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	.40	108.00	.02	1.08	42.50	27.10	-0.48	4.59	-18.90
Enwiki	4	1	.28	75.50	.03	.76	29.30	18.70	-0.49	4.65	-18.80
Ruwiki	4	1	.49	134.00	.05	1.34	52.40	33.40	-0.48	4.60	-18.90
Zhwiki	4	1	.19	51.30	.03	.52	20.20	12.90	-0.48	4.56	-18.80
								μ	**-0.48**	4.60	-18.90
								σ	0.01	0.03	0.05

Table D.148: `base64` PAPI comparison between `FB64` and `CURR` on `AVX2`

CONFIG			FB64			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	.47	52.50	.02	.78	22.60	22.40	-0.22	2.09	-15.60
Enwiki	4	1	.32	36.40	.02	.54	15.70	15.50	-0.22	2.09	-15.60
Ruwiki	4	1	.58	64.70	.03	.96	27.90	27.60	-0.22	2.09	-15.60
Zhwiki	4	1	.22	25.00	.02	.37	10.80	10.60	-0.22	2.09	-15.60
								μ	**-0.22**	2.09	-15.60
								σ	0.00	0.00	0.00

Table D.149: `base64` PAPI comparison between `FB64` and `CURR` on `AVX-512`

D.6.2 Case study: `csv2json` vs. sequential `csv2json`

CONFIG			SEQ			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Census	4	1	77.30	1.99	1.86	77.70	170.00	3.17	-0.13	-5.88	-0.46
Fin	4	1	.21	.01	.01	.50	1.22	.02	-22.70	-9.24	-1.17
Trade	4	1	3.74	.13	.12	7.97	18.30	.37	-18.10	-7.77	-1.07
								μ	**-13.70**	-7.63	-0.90
								σ	9.75	1.37	0.31

Table D.150: `csv2json` PAPI comparison between `SEQ` and `CURR` on `SSE-4.2`

CONFIG			SEQ			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Census	4	1	68.20	29.60	21.90	20.30	1180.00	17.50	16.80	-40.20	1.53
Fin	4	1	.20	.21	.05	.14	8.44	.12	4.52	-63.10	-5.69
Trade	4	1	3.16	2.84	2.04	2.12	128.00	1.68	4.44	-53.50	1.52
								μ	**8.58**	-52.30	-0.88
								σ	5.79	9.39	3.40

Table D.151: `csv2json` PAPI comparison between `SEQ` and `CURR` on `AVX2`

CONFIG			SEQ			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Census	4	1	68.60	13.50	13.40	12.00	33.80	13.40	19.80	-0.71	-0.00
Fin	4	1	.17	.09	.04	.09	.94	.09	6.73	-6.47	-4.57
Trade	4	1	2.78	1.30	1.24	1.29	11.70	1.29	6.36	-4.44	-0.22
								μ	**11.00**	-3.87	-1.60
								σ	6.28	2.38	2.10

Table D.152: `csv2json` PAPI comparison between `SEQ` and `CURR` on `AVX-512`

D.6.3 Case study: `icgrep` vs. `ugrep`

Arguments: `-colours=always`

CONFIG				UGREP			CURR			NORM. DIFFERENCE		
EXPRESSION	FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
@	Arwiki	4	1				7.62	11.90	5.82			
@	Arwiki	16	1	1.18	9.90	.02	7.32	16.80	7.79	-4.29	-0.48	-5.42
@	Enwiki	4	1	.81	6.85	.01	5.23	5.74	2.50	-4.45	0.11	-2.51
@	Enwiki	16	1	.81	6.92	.01	5.03	10.40	4.97	-4.25	-0.35	-5.00
@	Ruwiki	4	1				9.42	14.00	6.66			
@	Ruwiki	16	1	1.46	12.20	.02	9.04	20.30	9.34	-4.30	-0.46	-5.28
@	Zhwiki	4	1	.58	4.74	.01	3.66	5.30	2.43	-4.53	-0.08	-3.56
@	Zhwiki	16	1	.57	4.82	.01	3.52	8.13	3.63	-4.33	-0.49	-5.32
Date	Arwiki	4	1				9.61	18.00	4.92			
Date	Arwiki	16	1	4.73	2.92	.02	9.22	17.50	4.55	-3.14	-1.01	-3.16
Date	Enwiki	4	1	7.05	1.73	.01	5.65	3.82	1.84	1.41	-0.21	-1.85
Date	Enwiki	16	1	7.05	1.74	.01	5.43	4.70	2.58	1.63	-0.30	-2.59
Date	Ruwiki	4	1				12.10	23.60	5.91			
Date	Ruwiki	16	1	5.56	3.67	.01	11.60	22.60	5.45	-3.40	-1.07	-3.08
Date	Zhwiki	4	1	2.44	1.49	.00	4.93	10.40	2.31	-3.65	-1.30	-3.38
Date	Zhwiki	16	1	2.36	1.47	.01	4.70	10.20	2.17	-3.44	-1.28	-3.18
Email	Arwiki	4	1				13.80	13.80	2.83			
Email	Arwiki	16	1	648.00	3.80	.02	13.10	14.50	2.41	443.00	-0.75	-1.67
Email	Enwiki	4	1	551.00	2.64	.01	9.55	6.24	3.04	545.00	-0.36	-3.05
Email	Enwiki	16	1	551.00	2.53	.01	9.05	6.77	1.97	546.00	-0.43	-1.97
Email	Ruwiki	4	1				17.20	15.30	3.65			
Email	Ruwiki	16	1	653.00	4.77	.05	16.20	17.20	3.02	361.00	-0.70	-1.68
Email	Zhwiki	4	1	1100.00	2.25	.03	6.71	5.77	1.44	1,610.00	-0.52	-2.07
Email	Zhwiki	16	1	1100.00	2.10	.03	6.34	6.47	1.20	1,600.00	-0.64	-1.71
URIOrEmail	Arwiki	4	1				28.50	114.00	8.96			
URIOrEmail	Arwiki	16	1	389.00	7.01	.02	26.20	86.20	10.10	253.00	-5.53	-7.07
URIOrEmail	Enwiki	4	1	550.00	2.82	.02	10.40	6.49	2.84	543.00	-0.37	-2.84
URIOrEmail	Enwiki	16	1	550.00	2.86	.01	9.87	7.54	1.69	544.00	-0.47	-1.69
URIOrEmail	Ruwiki	4	1				35.10	135.00	10.30			
URIOrEmail	Ruwiki	16	1	489.00	8.96	.02	32.60	105.00	11.60	259.00	-5.44	-6.55
URIOrEmail	Zhwiki	4	1	885.00	3.84	.04	14.50	55.40	4.54	1,280.00	-7.56	-6.62
URIOrEmail	Zhwiki	16	1	881.00	3.69	.02	13.40	42.00	5.07	1,270.00	-5.62	-7.42
Xquote	Arwiki	4	1				32.10	117.00	9.02			
Xquote	Arwiki	16	1	8.51	4.26	.01	29.80	90.90	10.20	-14.80	-6.04	-7.11
Xquote	Enwiki	4	1	2.78	1.35	.02	13.00	9.33	2.25	-10.30	-0.80	-2.24
Xquote	Enwiki	16	1	2.77	1.38	.01	12.30	12.80	1.64	-9.64	-1.15	-1.65
Xquote	Ruwiki	4	1				37.50	128.00	9.54			
Xquote	Ruwiki	16	1	9.53	5.52	.01	34.90	103.00	10.70	-14.40	-5.53	-6.06
Xquote	Zhwiki	4	1	4.79	2.16	.02	15.40	51.50	4.27	-15.50	-7.25	-6.24
Xquote	Zhwiki	16	1	4.17	2.18	.00	14.30	41.70	4.77	-14.90	-5.80	-7.00
\X	Arwiki	4	1				98.60	410.00	31.60			
\X	Arwiki	16	1	546.00	10.30	.02	91.40	212.00	33.20	318.00	-14.10	-23.10
\X	Enwiki	4	1	144.00	3.28	.02	70.40	293.00	25.80	74.50	-29.20	-26.00
\X	Enwiki	16	1	140.00	3.13	.00	65.40	153.00	27.80	75.30	-15.10	-28.00
\X	Ruwiki	4	1				114.00	501.00	34.50			
\X	Ruwiki	16	1	902.00	14.20	.04	105.00	256.00	34.80	451.00	-13.70	-19.70
\X	Zhwiki	4	1	262.00	3.64	.01	66.70	236.00	15.10	286.00	-34.20	-22.10
\X	Zhwiki	16	1	259.00	3.53	.01	63.40	112.00	15.30	287.00	-15.90	-22.50
\p{Greek}	Arwiki	4	1				13.10	13.20	3.10			
\p{Greek}	Arwiki	16	1	4.01	2.44	.01	12.40	13.80	2.62	-5.88	-0.80	-1.82
\p{Greek}	Enwiki	4	1	2.79	1.65	.01	8.82	5.91	3.02	-6.08	-0.43	-3.03
\p{Greek}	Enwiki	16	1	2.79	1.63	.01	8.37	6.34	2.14	-5.62	-0.47	-2.15
\p{Greek}	Ruwiki	4	1				16.80	18.80	4.00			
\p{Greek}	Ruwiki	16	1	5.30	3.25	.01	16.00	20.10	3.43	-6.04	-0.95	-1.94
\p{Greek}	Zhwiki	4	1	2.07	1.27	.00	6.72	6.62	1.58	-6.84	-0.79	-2.31
\p{Greek}	Zhwiki	16	1	2.00	1.27	.00	6.41	7.02	1.37	-6.48	-0.84	-2.00
									μ	**252.00**	-4.48	-6.52
									σ	433.00	7.50	7.27

Table D.153: `icgrep -colours=always` PAPI comparison between `UGREP` and `CURR` on `SSE-4.2`

CONFIG				UGREP			CURR			NORM. DIFFERENCE		
EXPRESSION	FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
@	Arwiki	4	1	1.01	82.40	.01	2.63	54.60	23.00	-1.13	1.94	-16.00
@	Enwiki	4	1	.69	54.90	.01	1.80	30.50	15.90	-1.12	2.45	-16.00
@	Ruwiki	4	1	1.25	103.00	.01	3.28	60.80	28.30	-1.15	2.37	-16.00
@	Zhwiki	4	1	.49	35.30	.01	1.27	24.10	11.00	-1.15	1.65	-16.10
Date	Arwiki	4	1	5.78	69.70	.01	3.37	108.00	33.40	1.68	-2.66	-23.30
Date	Enwiki	4	1	6.85	66.00	.01	2.02	35.20	21.20	4.86	3.10	-21.40
Date	Ruwiki	4	1	6.95	80.60	.01	4.25	134.00	41.30	1.53	-3.03	-23.40
Date	Zhwiki	4	1	2.84	27.90	.01	1.71	58.00	16.20	1.66	-4.41	-23.80
Email	Arwiki	4	1	538.00	94.90	.03	4.65	78.40	31.70	372.00	1.15	-22.10
Email	Enwiki	4	1	441.00	64.50	.03	3.24	46.50	21.80	442.00	1.80	-22.00
Email	Ruwiki	4	1	539.00	115.00	.04	5.82	91.70	38.80	302.00	1.35	-22.00
Email	Zhwiki	4	1	862.00	44.40	.05	2.28	35.00	15.00	1,260.00	1.38	-22.00
URIOrEmail	Arwiki	4	1	322.00	61.10	.03	8.94	463.00	56.50	218.00	-28.10	-39.40
URIOrEmail	Enwiki	4	1	441.00	65.10	.02	3.57	50.20	21.90	441.00	1.50	-22.10
URIOrEmail	Ruwiki	4	1	399.00	75.90	.03	11.10	562.00	67.70	220.00	-27.50	-38.30
URIOrEmail	Zhwiki	4	1	685.00	32.00	.03	4.59	241.00	28.20	999.00	-30.70	-41.40
Xquote	Arwiki	4	1	7.19	52.60	.01	10.20	496.00	57.70	-2.12	-30.90	-40.20
Xquote	Enwiki	4	1	2.31	37.30	.01	4.46	68.00	22.10	-2.17	-3.09	-22.30
Xquote	Ruwiki	4	1	8.03	66.20	.02	12.10	556.00	65.60	-2.28	-27.70	-37.10
Xquote	Zhwiki	4	1	3.60	25.90	.01	5.00	229.00	27.60	-2.06	-29.80	-40.40
\X	Arwiki	4	1	477.00	56.80	.03	34.40	2390.00	340.00	309.00	-163.00	-237.00
\X	Enwiki	4	1	140.00	39.20	.02	24.40	1620.00	122.00	117.00	-159.00	-122.00
\X	Ruwiki	4	1	779.00	66.70	.05	39.10	2820.00	392.00	419.00	-156.00	-222.00
\X	Zhwiki	4	1	227.00	26.00	.02	23.00	1510.00	142.00	299.00	-218.00	-209.00
\p{Greek}	Arwiki	4	1	3.89	90.10	.01	4.43	74.50	31.60	-0.37	1.09	-22.00
\p{Greek}	Enwiki	4	1	2.70	53.80	.01	3.01	41.00	22.00	-0.31	1.30	-22.20
\p{Greek}	Ruwiki	4	1	5.07	96.80	.01	5.83	104.00	39.90	-0.43	-0.40	-22.60
\p{Greek}	Zhwiki	4	1	1.93	37.10	.01	2.37	36.70	15.30	-0.65	0.05	-22.40
									μ	**193.00**	-30.80	-50.20
									σ	307.00	60.40	63.00

Table D.154: `icgrep -colours=always` PAPI comparison between `UGREP` and `CURR` on `AVX2`

CONFIG				UGREP			CURR			NORM. DIFFERENCE		
EXPRESSION	FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
@	Arwiki	4	1				1.72	23.50	22.50			
@	Arwiki	16	1	.89	3.96	.01	1.55	23.40	22.60	-0.46	-1.36	-15.80
@	Enwiki	4	1	.61	.85	.01	1.20	15.90	15.60	-0.60	-1.52	-15.70
@	Enwiki	16	1	.61	2.19	.01	1.07	15.90	15.60	-0.46	-1.38	-15.70
@	Ruwiki	4	1				2.13	29.30	27.80			
@	Ruwiki	16	1	1.12	5.79	.01	1.92	29.10	27.80	-0.46	-1.32	-15.80
@	Zhwiki	4	1	.43	1.12	.02	.83	11.60	10.80	-0.58	-1.54	-15.80
@	Zhwiki	16	1	.43	1.71	.00	.76	11.60	10.80	-0.47	-1.45	-15.90
Date	Arwiki	4	1				2.17	29.70	24.50			
Date	Arwiki	16	1	4.41	5.11	.01	2.00	30.90	24.60	1.68	-1.80	-17.20
Date	Enwiki	4	1	7.02	2.59	.01	1.32	15.90	15.50	5.74	-1.34	-15.70
Date	Enwiki	16	1	7.02	2.65	.01	1.17	15.80	15.50	5.89	-1.33	-15.70
Date	Ruwiki	4	1				2.72	37.40	30.20			
Date	Ruwiki	16	1	5.20	6.36	.01	2.51	39.00	30.30	1.52	-1.85	-17.10
Date	Zhwiki	4	1	2.28	2.50	.01	1.10	15.10	11.90	1.73	-1.85	-17.40
Date	Zhwiki	16	1	2.19	2.21	.01	1.03	16.00	12.00	1.71	-2.03	-17.50
Email	Arwiki	4	1				2.84	23.30	22.50			
Email	Arwiki	16	1	565.00	8.58	.09	2.57	23.30	22.50	393.00	-1.02	-15.70
Email	Enwiki	4	1	465.00	4.84	.03	2.03	15.90	15.60	466.00	-1.12	-15.60
Email	Enwiki	16	1	465.00	4.34	.04	1.79	15.80	15.60	467.00	-1.16	-15.60
Email	Ruwiki	4	1				3.57	29.10	27.70			
Email	Ruwiki	16	1	564.00	8.21	.07	3.23	29.00	27.80	318.00	-1.18	-15.70
Email	Zhwiki	4	1	912.00	4.00	.10	1.39	11.50	10.80	1,340.00	-1.10	-15.70
Email	Zhwiki	16	1	912.00	3.85	.10	1.26	11.50	10.80	1,340.00	-1.12	-15.70
URIOrEmail	Arwiki	4	1				5.88	57.50	41.50			
URIOrEmail	Arwiki	16	1	339.00	8.50	.04	5.57	83.80	41.80	233.00	-5.25	-29.20
URIOrEmail	Enwiki	4	1	460.00	5.05	.03	2.17	15.90	15.60	462.00	-1.10	-15.60
URIOrEmail	Enwiki	16	1	460.00	4.53	.04	1.93	15.90	15.60	462.00	-1.14	-15.60
URIOrEmail	Ruwiki	4	1				7.28	66.80	49.50			
URIOrEmail	Ruwiki	16	1	425.00	11.50	.06	6.86	98.80	49.80	237.00	-4.95	-28.20
URIOrEmail	Zhwiki	4	1	726.00	4.72	.07	3.04	28.50	20.70	1,060.00	-3.49	-30.20
URIOrEmail	Zhwiki	16	1	725.00	5.44	.07	2.88	42.20	20.80	1,060.00	-5.39	-30.50
Xquote	Arwiki	4	1				6.57	66.50	42.10			
Xquote	Arwiki	16	1	7.98	5.38	.01	6.22	87.30	42.70	1.22	-5.71	-29.80
Xquote	Enwiki	4	1	2.64	2.33	.01	2.61	17.50	15.80	0.04	-1.53	-15.90
Xquote	Enwiki	16	1	2.63	2.26	.01	2.36	17.40	15.80	0.27	-1.53	-16.00
Xquote	Ruwiki	4	1				7.65	64.90	47.80			
Xquote	Ruwiki	16	1	8.98	6.65	.01	7.23	92.80	48.20	0.99	-4.88	-27.30
Xquote	Zhwiki	4	1	4.46	2.22	.01	3.23	30.80	20.10	1.80	-4.19	-29.50
Xquote	Zhwiki	16	1	3.93	2.38	.01	3.05	39.80	20.30	1.28	-5.50	-29.80
\X	Arwiki	4	1				25.10	246.00	113.00			
\X	Arwiki	16	1	478.00	8.78	.05	24.10	397.00	129.00	317.00	-27.10	-89.80
\X	Enwiki	4	1	134.00	4.66	.01	19.30	147.00	89.20	115.00	-14.30	-89.80
\X	Enwiki	16	1	133.00	4.49	.01	18.50	261.00	99.20	115.00	-25.80	-100.00
\X	Ruwiki	4	1				27.40	236.00	124.00			
\X	Ruwiki	16	1	786.00	11.50	.10	25.90	393.00	132.00	430.00	-21.60	-74.90
\X	Zhwiki	4	1	230.00	4.12	.02	15.50	108.00	53.00	314.00	-15.20	-77.80
\X	Zhwiki	16	1	226.00	3.79	.02	14.80	178.00	58.40	310.00	-25.60	-85.70
\p{Greek}	Arwiki	4	1				2.80	24.00	22.50			
\p{Greek}	Arwiki	16	1	3.73	4.19	.01	2.52	23.90	22.60	0.85	-1.38	-15.80
\p{Greek}	Enwiki	4	1	2.59	2.18	.01	1.98	16.20	15.60	0.62	-1.41	-15.70
\p{Greek}	Enwiki	16	1	2.59	2.45	.01	1.73	16.10	15.60	0.86	-1.38	-15.80
\p{Greek}	Ruwiki	4	1				3.75	32.80	28.30			
\p{Greek}	Ruwiki	16	1	4.90	5.76	.01	3.39	32.80	28.40	0.85	-1.53	-16.10
\p{Greek}	Zhwiki	4	1	1.87	1.82	.01	1.50	12.50	10.90	0.54	-1.57	-16.00
\p{Greek}	Zhwiki	16	1	1.85	2.01	.01	1.36	12.50	11.00	0.72	-1.54	-16.10
									μ	**225.00**	-4.97	-28.60
									σ	359.00	7.20	24.30

Table D.155: `icgrep -colours=always` PAPI comparison between `UGREP` and `CURR` on `AVX-512`

Arguments: `-colours=never`

CONFIG				UGREP			CURR			NORM. DIFFERENCE		
EXPRESSION	FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
@	Arwiki	4	1				1.30	11.80	5.13			
@	Arwiki	16	1	1.18	9.94	.03	1.21	12.80	5.93	-0.02	-0.20	-4.12
@	Enwiki	4	1	.81	6.82	.01	.89	7.89	3.36	-0.08	-0.11	-3.38
@	Enwiki	16	1	.81	6.84	.01	.83	8.66	3.97	-0.02	-0.18	-3.99
@	Ruwiki	4	1				1.62	14.60	6.23			
@	Ruwiki	16	1	1.46	12.20	.02	1.51	16.10	7.48	-0.03	-0.23	-4.23
@	Zhwiki	4	1	.61	4.49	.01	.62	5.34	2.19	-0.00	-0.12	-3.21
@	Zhwiki	16	1	.57	4.75	.01	.57	5.82	2.57	-0.00	-0.16	-3.76
Date	Arwiki	4	1				1.78	6.35	3.31			
Date	Arwiki	16	1	4.71	2.90	.01	1.69	5.97	3.01	2.11	-0.21	-2.09
Date	Enwiki	4	1	7.05	1.74	.01	1.17	3.21	1.72	5.93	-0.15	-1.73
Date	Enwiki	16	1	7.06	1.73	.01	1.11	2.86	1.48	5.99	-0.12	-1.49
Date	Ruwiki	4	1				2.21	7.92	4.07			
Date	Ruwiki	16	1	5.56	3.65	.01	2.10	7.33	3.66	1.96	-0.21	-2.07
Date	Zhwiki	4	1	2.44	1.49	.00	.86	3.06	1.59	2.32	-0.23	-2.33
Date	Zhwiki	16	1	2.33	1.48	.00	.82	2.81	1.44	2.23	-0.20	-2.10
Email	Arwiki	4	1				3.75	7.83	3.61			
Email	Arwiki	16	1	647.00	3.78	.04	3.62	8.18	3.84	449.00	-0.31	-2.65
Email	Enwiki	4	1	550.00	2.45	.01	1.80	5.23	2.40	553.00	-0.28	-2.41
Email	Enwiki	16	1	550.00	2.67	.03	1.71	5.42	2.52	553.00	-0.28	-2.50
Email	Ruwiki	4	1				4.94	9.60	4.34			
Email	Ruwiki	16	1	654.00	4.83	.04	4.76	10.10	4.64	368.00	-0.30	-2.60
Email	Zhwiki	4	1	1090.00	2.28	.04	2.23	3.68	1.67	1,600.00	-0.20	-2.39
Email	Zhwiki	16	1	1090.00	2.03	.03	2.16	3.89	1.80	1,600.00	-0.27	-2.60
URIOrEmail	Arwiki	4	1				3.79	16.10	7.74			
URIOrEmail	Arwiki	16	1	391.00	6.88	.01	3.66	16.40	7.74	270.00	-0.66	-5.39
URIOrEmail	Enwiki	4	1	550.00	2.64	.01	2.05	3.27	1.80	552.00	-0.06	-1.80
URIOrEmail	Enwiki	16	1	552.00	2.70	.02	1.96	3.05	1.48	554.00	-0.04	-1.47
URIOrEmail	Ruwiki	4	1				4.70	18.70	8.81			
URIOrEmail	Ruwiki	16	1	493.00	8.96	.04	4.53	18.80	8.74	277.00	-0.56	-4.93
URIOrEmail	Zhwiki	4	1	882.00	3.85	.02	1.94	8.05	3.84	1,290.00	-0.62	-5.60
URIOrEmail	Zhwiki	16	1	882.00	3.93	.01	1.87	8.10	3.81	1,290.00	-0.61	-5.58
Xquote	Arwiki	4	1				4.87	15.80	7.67			
Xquote	Arwiki	16	1	8.46	4.42	.01	4.70	16.00	7.73	2.62	-0.81	-5.38
Xquote	Enwiki	4	1	2.78	1.37	.01	2.76	3.41	1.65	0.02	-0.21	-1.66
Xquote	Enwiki	16	1	2.88	1.37	.01	2.65	3.23	1.53	0.23	-0.19	-1.54
Xquote	Ruwiki	4	1				6.03	17.50	8.33			
Xquote	Ruwiki	16	1	9.59	5.50	.01	5.79	17.80	8.40	2.15	-0.70	-4.75
Xquote	Zhwiki	4	1	4.79	2.17	.00	2.56	7.48	3.57	3.26	-0.78	-5.24
Xquote	Zhwiki	16	1	4.17	2.18	.00	2.48	7.57	3.61	2.48	-0.79	-5.29
\X	Arwiki	4	1				67.20	91.20	24.70			
\X	Arwiki	16	1	549.00	10.30	.03	64.90	76.60	26.80	338.00	-4.63	-18.70
\X	Enwiki	4	1	145.00	3.53	.00	45.60	55.10	14.60	101.00	-5.19	-14.70
\X	Enwiki	16	1	140.00	3.15	.01	44.20	47.30	16.30	97.00	-4.45	-16.40
\X	Ruwiki	4	1				81.10	94.10	27.60			
\X	Ruwiki	16	1	905.00	14.30	.07	78.10	86.40	30.70	468.00	-4.08	-17.40
\X	Zhwiki	4	1	266.00	3.39	.01	33.90	39.40	11.90	341.00	-5.29	-17.40
\X	Zhwiki	16	1	261.00	3.23	.01	33.10	36.80	13.00	334.00	-4.92	-19.10
\p{Greek}	Arwiki	4	1				2.42	4.70	2.29			
\p{Greek}	Arwiki	16	1	4.09	2.42	.01	2.27	4.47	2.08	1.27	-0.14	-1.44
\p{Greek}	Enwiki	4	1	2.79	1.64	.01	1.46	3.15	1.52	1.33	-0.15	-1.53
\p{Greek}	Enwiki	16	1	2.79	1.65	.01	1.36	2.96	1.38	1.44	-0.13	-1.39
\p{Greek}	Ruwiki	4	1				3.25	6.55	3.24			
\p{Greek}	Ruwiki	16	1	5.30	3.31	.01	3.04	6.29	2.99	1.28	-0.17	-1.69
\p{Greek}	Zhwiki	4	1	2.02	1.26	.00	1.49	2.41	1.21	0.78	-0.17	-1.77
\p{Greek}	Zhwiki	16	1	2.00	1.26	.00	1.40	2.27	1.09	0.88	-0.15	-1.60
									μ	**264.00**	-0.94	-5.03
									σ	432.00	1.58	5.20

Table D.156: `icgrep -colours=never` PAPI comparison between `UGREP` and `CURR` on `SSE-4.2`

CONFIG				UGREP			CURR			NORM. DIFFERENCE		
EXPRESSION	FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
@	Arwiki	4	1	1.01	82.80	.01	.78	39.40	22.40	0.16	3.02	-15.70
@	Enwiki	4	1	.69	55.50	.01	.53	27.20	15.50	0.16	2.85	-15.60
@	Ruwiki	4	1	1.25	103.00	.01	.96	48.60	27.60	0.16	3.07	-15.60
@	Zhwiki	4	1	.49	35.80	.01	.37	18.90	10.70	0.17	2.48	-15.70
Date	Arwiki	4	1	5.78	71.80	.01	1.03	47.20	32.10	3.31	1.72	-22.40
Date	Enwiki	4	1	6.85	65.30	.01	.65	29.90	21.10	6.25	3.57	-21.20
Date	Ruwiki	4	1	6.95	79.10	.01	1.28	58.30	39.40	3.21	1.18	-22.30
Date	Zhwiki	4	1	2.84	28.40	.01	.51	23.10	15.40	3.42	0.78	-22.60
Email	Arwiki	4	1	536.00	94.70	.05	2.04	40.20	24.90	373.00	3.80	-17.40
Email	Enwiki	4	1	441.00	64.40	.03	1.10	28.10	17.10	443.00	3.66	-17.20
Email	Ruwiki	4	1	537.00	116.00	.05	2.74	50.20	30.40	303.00	3.72	-17.20
Email	Zhwiki	4	1	863.00	44.20	.06	1.22	19.60	11.80	1,270.00	3.61	-17.20
URIOrEmail	Arwiki	4	1	321.00	61.00	.02	2.67	74.10	47.50	222.00	-0.91	-33.10
URIOrEmail	Enwiki	4	1	441.00	64.60	.03	1.23	30.80	22.10	444.00	3.41	-22.20
URIOrEmail	Ruwiki	4	1	399.00	75.70	.03	3.25	88.00	56.50	224.00	-0.70	-32.00
URIOrEmail	Zhwiki	4	1	686.00	32.30	.03	1.35	36.50	23.00	1,000.00	-0.61	-33.80
Xquote	Arwiki	4	1	7.20	52.80	.01	3.26	73.80	47.00	2.75	-1.47	-32.80
Xquote	Enwiki	4	1	2.30	37.20	.01	1.67	31.70	22.10	0.64	0.55	-22.20
Xquote	Ruwiki	4	1	8.03	66.50	.02	3.99	86.30	55.20	2.29	-1.12	-31.30
Xquote	Zhwiki	4	1	3.59	25.60	.01	1.70	35.30	22.20	2.78	-1.42	-32.50
\X	Arwiki	4	1	477.00	56.00	.03	23.90	373.00	98.00	316.00	-22.10	-68.40
\X	Enwiki	4	1	141.00	39.40	.02	16.50	241.00	60.10	125.00	-20.30	-60.50
\X	Ruwiki	4	1	779.00	66.90	.04	28.60	474.00	112.00	425.00	-23.00	-63.40
\X	Zhwiki	4	1	227.00	25.50	.02	12.70	183.00	47.20	315.00	-23.10	-69.30
\p{Greek}	Arwiki	4	1	3.89	90.20	.01	1.42	45.20	32.00	1.73	3.14	-22.30
\p{Greek}	Enwiki	4	1	2.70	54.10	.01	.95	31.50	22.10	1.76	2.27	-22.30
\p{Greek}	Ruwiki	4	1	5.07	97.30	.01	1.98	57.20	39.60	1.75	2.27	-22.40
\p{Greek}	Zhwiki	4	1	1.93	37.30	.01	.87	22.00	15.20	1.55	2.26	-22.30
									μ	**196.00**	-1.70	-29.00
									σ	308.00	8.53	16.00

Table D.157: `icgrep -colours=never` PAPI comparison between `UGREP` and `CURR` on `AVX2`

CONFIG				UGREP			CURR			NORM. DIFFERENCE		
EXPRESSION	FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
@	Arwiki	4	1				.64	22.80	22.40			
@	Arwiki	16	1	.89	3.53	.01	.59	22.80	22.40	0.21	-1.35	-15.60
@	Enwiki	4	1	.61	2.07	.01	.44	15.80	15.50	0.17	-1.38	-15.60
@	Enwiki	16	1	.61	1.95	.01	.41	15.80	15.50	0.20	-1.39	-15.60
@	Ruwiki	4	1				.79	28.10	27.60			
@	Ruwiki	16	1	1.11	4.76	.01	.73	28.20	27.60	0.21	-1.33	-15.60
@	Zhwiki	4	1	.44	1.31	.01	.31	10.90	10.70	0.19	-1.41	-15.70
@	Zhwiki	16	1	.43	1.77	.00	.28	10.90	10.70	0.22	-1.34	-15.70
Date	Arwiki	4	1				.80	24.70	23.70			
Date	Arwiki	16	1	4.41	5.40	.01	.73	24.70	23.70	2.57	-1.35	-16.60
Date	Enwiki	4	1	7.02	1.45	.01	.48	15.80	15.50	6.59	-1.44	-15.60
Date	Enwiki	16	1	7.02	2.02	.01	.43	15.80	15.50	6.64	-1.39	-15.60
Date	Ruwiki	4	1				.98	30.30	29.20			
Date	Ruwiki	16	1	5.20	8.07	.01	.90	30.30	29.20	2.43	-1.26	-16.50
Date	Zhwiki	4	1	2.28	1.65	.01	.39	11.90	11.30	2.77	-1.51	-16.60
Date	Zhwiki	16	1	2.19	2.34	.01	.36	11.90	11.30	2.68	-1.41	-16.60
Email	Arwiki	4	1				1.54	22.90	22.40			
Email	Arwiki	16	1	564.00	7.08	.07	1.48	23.00	22.40	393.00	-1.11	-15.60
Email	Enwiki	4	1	465.00	4.34	.03	.91	15.90	15.50	467.00	-1.17	-15.60
Email	Enwiki	16	1	465.00	4.19	.03	.86	15.90	15.50	467.00	-1.18	-15.60
Email	Ruwiki	4	1				2.11	28.30	27.60			
Email	Ruwiki	16	1	567.00	8.70	.07	2.03	28.40	27.60	320.00	-1.11	-15.60
Email	Zhwiki	4	1	912.00	3.67	.09	.92	11.00	10.70	1,340.00	-1.07	-15.50
Email	Zhwiki	16	1	912.00	4.34	.09	.89	11.00	10.70	1,340.00	-0.98	-15.50
URIOrEmail	Arwiki	4	1				2.20	38.70	35.40			
URIOrEmail	Arwiki	16	1	340.00	8.86	.03	2.12	38.70	35.40	235.00	-2.08	-24.70
URIOrEmail	Enwiki	4	1	460.00	4.71	.05	.84	15.90	15.50	463.00	-1.13	-15.60
URIOrEmail	Enwiki	16	1	460.00	4.32	.04	.78	15.90	15.50	463.00	-1.16	-15.60
URIOrEmail	Ruwiki	4	1				2.64	45.60	42.10			
URIOrEmail	Ruwiki	16	1	425.00	11.20	.05	2.54	45.60	42.10	239.00	-1.95	-23.80
URIOrEmail	Zhwiki	4	1	726.00	4.92	.05	1.11	19.00	17.20	1,060.00	-2.07	-25.20
URIOrEmail	Zhwiki	16	1	725.00	4.89	.05	1.07	19.00	17.20	1,060.00	-2.07	-25.20
Xquote	Arwiki	4	1				2.56	38.20	35.00			
Xquote	Arwiki	16	1	7.98	5.48	.01	2.47	38.10	35.00	3.84	-2.28	-24.40
Xquote	Enwiki	4	1	2.64	2.22	.01	1.12	16.10	15.60	1.53	-1.40	-15.70
Xquote	Enwiki	16	1	2.63	2.23	.01	1.05	16.00	15.60	1.59	-1.39	-15.70
Xquote	Ruwiki	4	1				3.06	44.20	40.80			
Xquote	Ruwiki	16	1	8.99	6.73	.03	2.95	44.00	40.80	3.42	-2.11	-23.10
Xquote	Zhwiki	4	1	4.46	2.26	.01	1.31	18.20	16.60	4.62	-2.35	-24.40
Xquote	Zhwiki	16	1	3.93	2.20	.02	1.27	18.20	16.60	3.91	-2.34	-24.30
\X	Arwiki	4	1				20.30	92.20	77.90			
\X	Arwiki	16	1	479.00	8.80	.06	19.80	85.10	77.90	320.00	-5.33	-54.40
\X	Enwiki	4	1	134.00	4.18	.03	14.40	58.40	47.40	120.00	-5.47	-47.80
\X	Enwiki	16	1	133.00	4.40	.01	13.90	56.30	47.40	120.00	-5.23	-47.80
\X	Ruwiki	4	1				23.90	112.00	88.40			
\X	Ruwiki	16	1	785.00	11.80	.12	23.10	97.70	88.40	432.00	-4.86	-50.00
\X	Zhwiki	4	1	230.00	4.02	.03	10.50	46.00	37.50	322.00	-6.16	-55.10
\X	Zhwiki	16	1	226.00	4.17	.03	10.20	42.70	37.60	317.00	-5.65	-55.10
\p{Greek}	Arwiki	4	1				1.12	23.10	22.40			
\p{Greek}	Arwiki	16	1	3.73	4.20	.01	1.04	23.10	22.40	1.88	-1.32	-15.60
\p{Greek}	Enwiki	4	1	2.59	2.71	.01	.79	16.00	15.50	1.81	-1.34	-15.60
\p{Greek}	Enwiki	16	1	2.59	2.28	.01	.73	15.90	15.50	1.88	-1.38	-15.60
\p{Greek}	Ruwiki	4	1				1.59	29.00	28.00			
\p{Greek}	Ruwiki	16	1	4.90	5.65	.01	1.48	28.90	27.90	1.94	-1.32	-15.80
\p{Greek}	Zhwiki	4	1	1.87	1.91	.01	.67	11.10	10.70	1.77	-1.35	-15.70
\p{Greek}	Zhwiki	16	1	1.85	1.97	.01	.62	11.10	10.70	1.81	-1.34	-15.70
									μ	**227.00**	-2.05	-22.50
									σ	360.00	1.44	12.40

Table D.158: `icgrep -colours=never` PAPI comparison between `UGREP` and `CURR` on `AVX-512`

D.6.4 Case study: `u8to16` vs. `utf8lut`

CONFIG			UTF8LUT			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	4.31	12.20	2.63	6.60	5.93	3.05	-1.60	0.44	-0.29
Enwiki	4	1	3.54	3.06	1.38	4.68	4.03	2.15	-1.15	-0.10	-0.78
Ruwiki	4	1	5.11	25.80	5.17	8.34	7.16	3.64	-1.83	1.06	0.86
Zhwiki	4	1	1.91	5.62	1.41	3.28	2.69	1.36	-2.01	0.43	0.07
								μ	**-1.65**	0.46	-0.03
								σ	0.32	0.41	0.60

Table D.159: `u8u16` PAPI comparison between `UTF8LUT` and `CURR` on `SSE-4.2`

CONFIG			UTF8LUT			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	4.30	125.00	37.20	5.09	57.50	31.60	-0.55	4.74	3.90
Enwiki	4	1	3.34	42.90	25.90	3.60	37.80	22.00	-0.26	0.52	3.91
Ruwiki	4	1	5.06	164.00	46.60	6.38	69.00	39.40	-0.75	5.39	4.08
Zhwiki	4	1	1.83	46.20	17.80	2.47	26.80	15.10	-0.94	2.85	4.06
								μ	**-0.62**	3.37	3.98
								σ	0.25	1.90	0.08

Table D.160: `u8u16` PAPI comparison between `UTF8LUT` and `CURR` on `AVX2`

CONFIG			UTF8LUT			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	4	1	3.97	44.20	22.50	2.80	22.90	22.40	0.81	1.48	0.11
Enwiki	4	1	3.23	18.20	15.70	1.99	15.90	15.50	1.25	0.23	0.18
Ruwiki	4	1	4.66	55.90	27.90	3.50	28.20	27.60	0.66	1.57	0.15
Zhwiki	4	1	1.74	16.30	10.80	1.35	10.90	10.60	0.57	0.79	0.22
								μ	**0.82**	1.02	0.17
								σ	0.26	0.55	0.04

Table D.161: `u8u16` PAPI comparison between `UTF8LUT` and `CURR` on `AVX-512`

D.6.5 Case study: `u32to8` vs. `utf8lut`

CONFIG			UTF8LUT			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	32	1	2.85	8.28	2.96	29.00	99.60	16.20	-6.29	-2.19	-3.19
Enwiki	32	1	2.62	7.40	2.68	21.70	89.20	16.60	-4.81	-2.07	-3.52
Ruwiki	32	1	3.16	9.35	3.26	34.60	119.00	17.60	-6.93	-2.40	-3.15
Zhwiki	32	1	1.17	3.62	1.13	13.30	46.20	6.74	-7.03	-2.46	-3.24
								μ	**-6.26**	-2.28	-3.27
								σ	0.89	0.16	0.15

Table D.162: `u32u8` PAPI comparison between `UTF8LUT` and `CURR` on `SSE-4.2`

CONFIG			UTF8LUT			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	32	1	2.85	143.00	101.00	8.02	645.00	79.10	-1.24	-12.10	5.29
Enwiki	32	1	2.65	137.00	96.30	6.30	509.00	75.50	-0.92	-9.40	5.26
Ruwiki	32	1	3.15	158.00	111.00	9.38	776.00	86.50	-1.37	-13.60	5.32
Zhwiki	32	1	1.20	59.80	42.00	3.60	299.00	32.90	-1.39	-13.80	5.26
								$\boldsymbol{\mu}$	**-1.23**	-12.20	5.28
								$\boldsymbol{\sigma}$	0.19	1.77	0.02

Table D.163: `u32u8` PAPI comparison between `UTF8LUT` and `CURR` on `AVX2`

CONFIG			UTF8LUT			CURR			NORM. DIFFERENCE		
FILE	SL	TC	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC ($\times 10^9$)	L2M ($\times 10^6$)	L3M ($\times 10^6$)	CYC	L2M ($\times 10^{-2}$)	L3M ($\times 10^{-3}$)
Arwiki	32	1	3.02	70.60	65.20	5.46	189.00	65.30	-0.58	-2.85	-0.02
Enwiki	32	1	2.83	65.10	62.00	4.30	159.00	62.00	-0.37	-2.36	0.00
Ruwiki	32	1	3.32	77.60	71.10	6.36	218.00	71.10	-0.67	-3.09	-0.01
Zhwiki	32	1	1.25	29.20	27.10	2.45	84.00	27.10	-0.69	-3.17	-0.04
								$\boldsymbol{\mu}$	**-0.58**	-2.87	-0.02
								$\boldsymbol{\sigma}$	0.13	0.32	0.02

Table D.164: `u32u8` PAPI comparison between `UTF8LUT` and `CURR` on `AVX-512`