EXPERIMENTAL ANALYSIS OF LAN SORTING ALGORITHMS

by

Mohamed Salehmohamed

B.Sc., Simon Fraser University, 1983

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the School

of

Computing Science

SIMON FRASER UNIVERSITY

April 1987

Approval

Name : Mohamed Salehmohamed

Degree : Master of Science

Title of Thesis : Experimental Analysis of LAN Sorting Algorithms

Examining Committee:

Chairman: Dr. Louis Hafer

Dr. Wo-Shun Luk
Senior Supervisor

Dr. Joseph G. Peters
Senior Supervisor

Dr. Arthur L. Liestman

Dr. Anthony H. Dixon
External Examiner
School of Computing Science
Simon Fraser University

13/4/87
Date Approved

Abstract

This thesis provides empirical results for selected parallel sorting algorithms (block sorting algorithms) and distributed sorting algorithms which have been adapted for implementation on an Ethernet network with diskless Sun workstations.

Most work concerning the performance of parallel and distributed sorting algorithms has been theoretical and assumes simplified models. Hence, we adopt an empirical approach which provides more insight into the performance of the algorithms. Our cost model considers both local processing costs and communication costs to be important factors when evaluating the performance of the sorting algorithms in the LAN environment.

We obtain our experimental results on communication time, local processing time and response time of each algorithm for various file sizes and different numbers of processors. These results are analyzed and compared to our theoretical model. In cases where the experimental results do not agree with the theoretical results, the discrepancies are explained. We also make an attempt to project the behaviour of the algorithms as number of processors or interprocess communication facilities changes.

To my wife, Zeinul

Acknowledgements

I would like to express my sincere gratitude to Dr. Wo-Shun Luk and Dr. Joseph Peters for their invaluable support and encouragement throughout my research. Special thanks to Dr. Arthur Liestman and Dr. Anthony Dixon for their advice on the revision of this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1. Network Configuration

Processors are interconnected mainly to share resources. There are a number of ways to interconnect them. One extreme is where large computers located at different geographic sites are linked together to form a *long-haul network*, e.g. ARPANET. The other extreme is where several computers are closely connected together to form a *multiprocessor system*. Near the middle of these extremes is local area networking (LAN), the interconnection of computers to gain the resource sharing of computer networking and the parallelism of multiprocessing [BrH 83, MeB 76].

The distance between computers and the associated communication data rate can be used to distinguish among the different methods of connecting processors (see table 1-1) [MeB 76].

System	Distance	Data Rate
Long-haul network	> 25 km.	$< .1$ Mbps
Local area network	25-.1 km.	.1-100 Mbps
Multiprocessors	$< .1$ km.	> 100 Mbps

Table 1-1: Characteristics of Distributed Processing Systems

Another distinction between local area networks and long-haul networks is that local area networks generally experience significantly fewer data transmission errors and much lower communication costs than long-haul networks, so cost-performance tradeoffs are very different [Sta 84].

The main difference between local area networks and multiprocessor systems is the degree of coupling. Multiprocessor systems are tightly coupled, and usually have centralized control, shared memory, and completely integrated communications functions. Local area networks tend to exhibit the opposite characteristics [Sta 84].

Local networks have become more popular in recent years. The main reasons are the continuing decrease in cost and an increase in the capabilities of computer hardware and local networking technology. There has been an increase in the use of systems consisting of single-user machines (workstations) interconnected by a fast local area network. The workstations usually have their own processor and memory, but they need to share other expensive resources such as disk storage and printers. The workstations can operate independently, or, if the application requires distributed processing, they communicate through the network. These systems are being used for many applications -- general office tasks, computer-aided engineering design, academic computing facilities, and software development, to name a few [Sta 84, Svo 84].

1.2. Parallel vs Distributed Sorting Algorithms

Sorting is theoretically interesting and an important application [Knu 73, Baa 78]. Over the past two decades, much computing science research has focussed on sorting on a single processor or on an array of processors. Recently, parallel sorting and distributed sorting have received increasing attention from computing science researchers. Parallel sorting algorithms are generally designed for a multiprocessor system whereas distributed sorting algorithms are generally designed for a network of computers.

The designer of a parallel sorting algorithm usually assumes that the number of processors available to perform the sort is very large. However, for a general-purpose sorting algorithm, it is desirable to set a limit on the number of processors available. We are, thus, interested in **Block sorting algorithms**. Block sorting algorithms require relatively small number of processors to sort a large array of keys. Also, block sorting algorithms can easily be adapted to the LAN environment.

Block sorting algorithms partition the file to be sorted into a number of blocks depending on the number of processors available to perform the sort. In the literature on parallel sorting algorithms, the time complexity is expressed in terms of parallel comparisons and exchanges between processors in the interconnecting network, i.e. local processing cost. The communication cost is assumed to be proportional to local processing cost and is usually ignored in the analysis of a parallel sorting algorithm.

In contrast, distributed sorting algorithms are designed for a network of processors that do not share memory. Here, fragments of the file to be sorted reside in the memory of each processor. Most distributed algorithm research assumes an environment in which communication costs are orders of magnitude higher than local processing costs. Consequently, communication costs dominate the sorting time and local processing costs are ignored.

1.3. Objectives of the Thesis

Most work concerning the performance of parallel and distributed sorting algorithms has been theoretical and assumes simplified models. In most research papers concerning this subject, empirical results are not provided. In the case of parallel sorting algorithms, the time complexity of the algorithm is usually given in terms of parallel comparisons. In the case of distributed sorting algorithms, the time complexity of the algorithm is usually given in terms of the number of messages. This thesis provides empirical results for selected parallel and distributed sorting algorithms which have been run in a local area network environment.

It is our belief that the cost models commonly used to evaluate parallel and distributed algorithms are not appropriate for distributed processing in a LAN environment. Since local area networks lie between multiprocessor systems and long-haul networks, our cost model considers both communication costs and local processing costs to be important factors when evaluating the efficiency of a sorting algorithm. To test this model, we have selected existing parallel and distributed sorting algorithms from the literature. These algorithms have been analyzed, adapted to our LAN environment and then implemented. These algorithms are called **Local Area Network Sorting Algorithms (LANSAs)**.

The goal of this thesis is to test the validity of our theoretical model. We take the experimental approach to determine how communication cost and local processing cost affect the performance of the algorithms. The experimental approach aids us in ranking the algorithms in ways that a theoretical analysis cannot. We believe that it also informs us more about the subtle behaviour of the LAN than a theoretical analysis could. As a consequence, the sorting algorithms may be improved due to the experimental observations.

The results gathered from our experiments are analyzed and compared to our theoretical model. In cases where the experimental results do not agree with the theoretical results, we propose explanations for the discrepancies. The algorithms are also ranked according to their performance.

1.4. Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 describes the model that we use in this thesis and the five LANSAs that we have implemented. Chapter 3 contains a theoretical cost analysis of the LANSAs according to our model. The experimental setup is described in Chapter 4 and the experimental results are presented and analyzed. Chapter 5 summarizes the major contributions of this thesis.

Chapter 2

LANSAs

In this chapter, we describe the model that we use in this thesis and the five LANSAs that we have implemented.

2.1. Model

Our model is a *broadcast network* consisting of a medium-speed bus (e.g. Ethernet) and p processors P_i, $1 \leqslant i \leqslant p$ which are attached to the bus. Processors can communicate with each other by sending messages through the network. A long message may be segmented into a number of *packets* that can be individually transmitted through the network.

When p processors are available and N keys are to be sorted, the keys are assumed to be distributed among the p processors so that a block of $M = \lfloor N/p \rfloor$ keys is stored in each processor's local memory. These blocks are sometimes referred to as local files. Processors are labeled $P_1, P_2, \ldots, P_p$ according to a presumed order. The processors cooperate to redistribute the keys so that the block residing in each processor's local memory is a sorted sequence of length approximately M, and the concatenation of these blocks (according to the presumed order) is a sorted sequence of length N. Processors do not share memory and each processor has sufficient memory to perform an internal sort of N keys.

In our theoretical cost model for the LAN environment, we consider two major components: communication cost and local processing cost. In our theoretical model, the number of parallel packet transmissions in the network is a lower bound on communication cost while the total number of packets transmitted in the network is an upper bound. A parallel packet transmission is the transmission of packets that are initiated simultaneously by the processors in the network. The local processing cost is the number of parallel comparison-exchanges performed by the processors. Experimentally, communication cost is the total time that network resources are being used. Local processing cost is the total time a processor takes to process a local activity. In both cases, the sum of communication cost and local processing cost yields the **response time** to solve the sorting problem.

2.2. LANSA 0

LANSA 0 is a **Centralized Sorting Algorithm**. Let a file, F, be a sequence of keys distributed equally among processors $P_1, P_2, \ldots, P_p$. Processors P_i, $2 \leqslant i \leqslant p$ transmit their local files to processor P_1. Processor P_1 sorts the combined files using the quicksort algorithm and redistributes the file equally so that each processor receives a locally sorted file which is also globally sorted.

2.3. LANSA 1

LANSA 1 has been adapted from the parallel block sorting algorithm, **Block Odd-Even Sort based on Two-Way Merge-Split** [BDHM 84]. Before the description of the algorithm is given, let us define a two-way merge-split step. A **two-way merge-split** step is defined as a two-way merge of two sorted blocks of size *M*, followed by a split of the resulting block of size 2*M* into two halves. Both operations are executed within a processor's local memory. The contents of a processor's memory before and after a two-way merge-split step are shown in Figure 2-1 [BDHM 84].

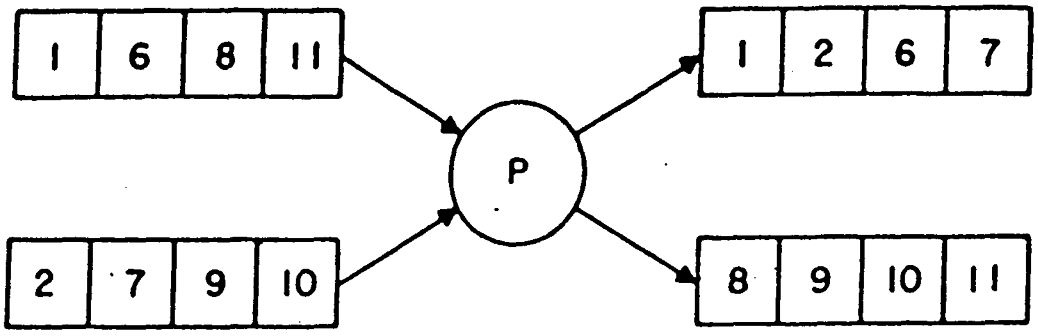

Figure 2-1: Two-Way Merge-Split step

LANSA 1 has been derived from the *Odd-Even Transposition Sort*, which is described in Appendix A. The algorithm consists of a preprocessing step (step 0) and *p* additional steps (steps 1 to *p*) where *p* is the number of processors. Initially, each processor's memory contains a sequence of length *M* (block size).

During step 0, each processor independently sorts the sequence residing in its local memory. This local sort step uses a quicksort algorithm. During steps 1 to p, all processors cooperate to merge the p sequences generated by step 0. These p steps are similar to the p steps of the odd-even transposition sort. During the odd (even) steps, the odd- (even-) numbered processors receive from their higher-even (higher-odd) numbered neighbor a sorted block, perform a two-way merge, and send back the higher M keys. During the odd- (even-) numbered steps, the odd- (even-) numbered processors are active while the even- (odd-) numbered processors are idle. This algorithm is illustrated in Figure 2-2 for four processors, where $M=5$.

2.4. LANSA 2

LANSA 2 has been adapted from the parallel block sorting algorithm, **Block Bitonic Sort based on Two-Way Merge-Split** [BDHM 84]. This algorithm also performs a two-way merge-split which has been defined in Section 2.3.

LANSA 2 has been derived from *Stone's Bitonic Sort*, which is described in Appendix A. The comparison-exchange step in Stone's bitonic sort is replaced by a two-way merge-split step to obtain LANSA 2. LANSA 2 can sort $M \cdot p$ keys with p processors in two local sort steps, $((\log p)(\log p+1)/2+1)$ shuffle steps (shuffle step is explained in Appendix A), $(\log p+1)(\log p+2)/2$ merge-split steps and $(\log p-1)$ transmission steps. The local sort steps use a quicksort algorithm. During a shuffle step, each processor sends a sorted sequence of length $M/2$ to

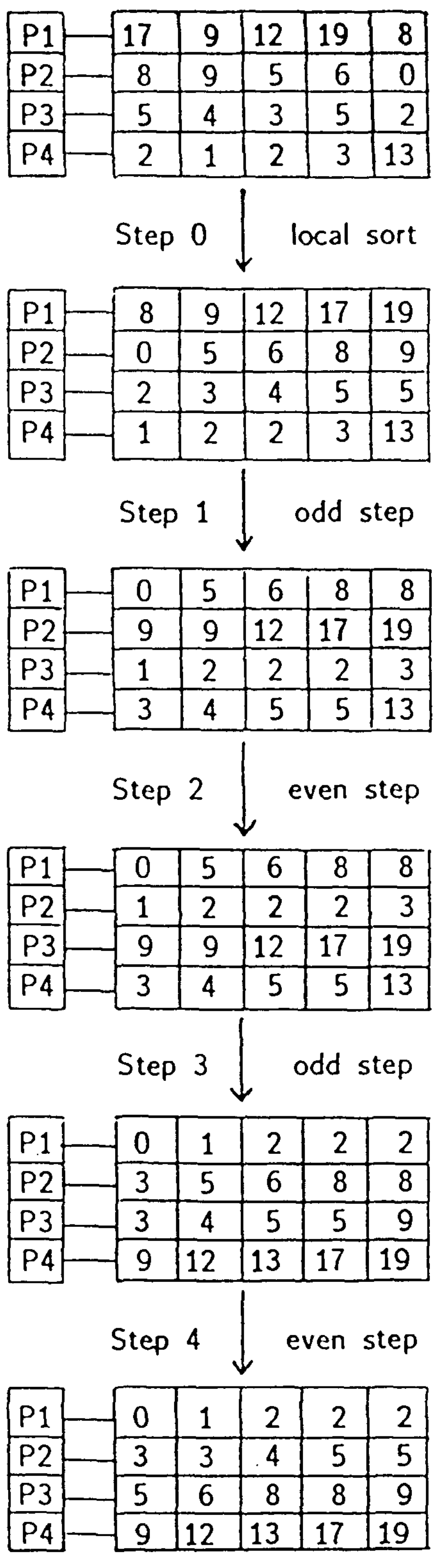

Figure 2-2: LANSA 1

each of its logical neighbors. During a merge-split step, each processor performs a two-way merge of the two sequences of length $M/2$ and splits the resulting sequence into two sequences of length $M/2$. The algorithm is illustrated in Figure 2-3 for two processors, where $M=4$ [BDHM-84].

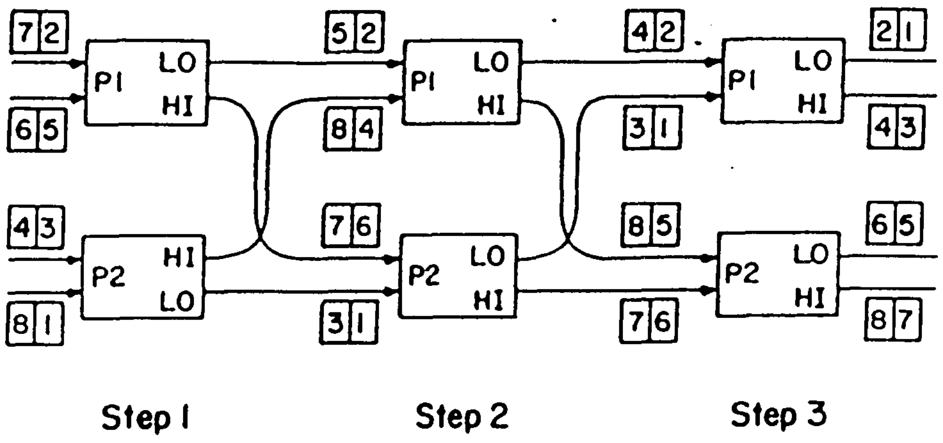

Figure 2-3: LANSA 2

LANSA 2 is different from Block Bitonic Sort based on Two-Way Merge-Split in that the transmission step is an added feature. The transmission steps reduce the number of shuffle steps. Each transmission step is the combination of several consecutive shuffle steps. There may be several transmission steps depending on the number of sets of consecutive shuffle steps. In our theoretical model and in our experimental broadcast network, data can be transmitted to any processor in the network in a single transmission step. This modification reduces a number of shuffle steps and makes LANSA 2 a faster algorithm than Block Bitonic Sort based on Two-Way Merge-Split.

2.5. LANSA 3

LANSA 3 has been adapted from the **Distributed Sorting** algorithm by [RSS 85]. Let F be a file of $M{\cdot}p$ keys distributed among p processors $P_1, P_2, \ldots, P_p$. Let $F[k]$ denote the k^{th} smallest element in F where $k = iM$, $1 \leqslant i < p$.

The algorithm consists of four phases. In the first phase, each processor locally sorts the sequence residing in its local memory using a quicksort algorithm. In the second phase, each key in F is assigned a destination. This is accomplished by determining the $F[k]$'s distributively. To find $F[k]$ distributively, the distributed selection algorithms in [SaS 83] and [SaS 82] are used. Once the $F[k]$'s are found, all processors assign destinations simultaneously. In the third phase, the keys are sent to their assigned destinations. This phase is called the **Routing phase**. In the final phase, each processor independently carries out a local sort using the quicksort algorithm. Figure 2-4 illustrates LANSA 3 for four processors, where $M = 5$. Note that in the example, the numbers in parenthesis refer to the processor number to which the key is assigned.

The distributed selection algorithms from [SaS 83] and [SaS 82] are described below. These algorithms have also been adapted to our model. We call these algorithms SEL 1 and SEL 2 respectively.

SEL 1 is a reduction technique for selection in distributed files. Let F be

P1	4	11	20	6	8
P2	19	17	3	7	9
P3	2	5	13	16	12
P4	10	1	18	15	14

Phase 1 (local sort)

P1	4	6	8	11	20
P2	3	7	9	17	19
P3	2	5	12	13	16
P4	1	10	14	15	18

Phase 2 (assign destination)

F[k]=5

P1	4(1)	6	8	11	20
P2	3(1)	7	9	17	19
P3	2(1)	5(1)	12	13	16
P4	1(1)	10	14	15	18

F[k]=10

P1	4(1)	6(2)	8(2)	11	20
P2	3(1)	7(2)	9(2)	17	19
P3	2(1)	5(1)	12	13	16
P4	1(1)	10(2)	14	15	18

F[k]=15

P1	4(1)	6(2)	8(2)	11(3)	20
P2	3(1)	7(2)	9(2)	17	19
P3	2(1)	5(1)	12(3)	13(3)	16
P4	1(1)	10(2)	14(3)	15(3)	18

P1	4(1)	6(2)	8(2)	11(3)	20(4)
P2	3(1)	7(2)	9(2)	17(4)	19(4)
P3	2(1)	5(1)	12(3)	13(3)	16(4)
P4	1(1)	10(2)	14(3)	15(3)	18(4)

Phase 3 (Routing phase)

P1	4	3	2	5	1
P2	7	9	6	8	10
P3	12	13	11	14	15
P4	18	20	17	19	16

Phase 4 (local sort)

P1	1	2	3	4	5
P2	6	7	8	9	10
P3	11	12	13	14	15
P4	16	17	18	19	20

Figure 2-4: LANSA 3

a file of N elements distributed among p processors. We want to find the k^{th} smallest element ($F[k]$) of F. SEL 1 is designed for applications in which the size of the file is much greater than the number of processors, i.e. $N>>p$. It goes through a sequence of iterations whose effect is to reduce the size of the problem until another efficient selection algorithm (SEL 2, for example) can be employed to determine $F[k]$. Occasionally, SEL 1 may locate the element being sought.

Since SEL 1 is a reduction algorithm, a predetermined parameter T (termination criterion) is used. When the size of the problem is reduced to a value smaller than T, the algorithm is terminated. We have determined T experimentally. SEL 1 also requires a controller to perform some bookkeeping tasks; we chose P_1 as the controller. The algorithm consists of four steps:

1. The controller, P_1, determines the current set size m_i of each P_i and finds the lowest numbered processor P_l with maximum set size m.

2. P_1 requests P_l to return its $\lfloor m k/n \rfloor^{th}$ value where $n=\Sigma_{i=1}^{p} m_i$. We call this value x.

3. P_1 broadcasts x. Each P_j $2 \leqslant j \leqslant p$, determines α_j and β_j, where α_j is the number of keys $< x$ and β_j is the number of keys $\leqslant x$ for processor P_j and sends both values to P_1. P_1, in the meantime, also determines α_1 and β_1.

4. Upon reception of all such values, P_1 determines $\alpha=\Sigma_{i=1}^{p} \alpha_i$ and $\beta=\Sigma_{i=1}^{p} \beta_i$ and

a. If $\alpha < k \leqslant \beta$, halt; x is the value sought.

b. If $\beta < k$, discard values $\leqslant x$; $k \leftarrow k - \beta$.

   If the stopping criterion is met, terminate the algorithm; otherwise return to step 1.

c. If $\beta > k$, discard values $> x$. Also discard value x at P_l.

   If the stopping criterion is met, terminate the algorithm; otherwise return to step 1.

SEL 2 is a distributed selection algorithm to find the k^{th} smallest element, $F[k]$, in a file of N elements distributed over p processors. Without loss of generality, let $k \leqslant \lfloor N/2 \rfloor$. The algorithm requires a controller to perform some bookkeeping tasks; we chose P_1 as the controller. SEL 2 consists of three steps:

1. The controller, P_1, requests each processor P_j, $2 \leqslant j \leqslant p$ to return its α^{th} smallest element, s_j where $\alpha = \lfloor (k-1)/p \rfloor$. P_1, in the meantime, also determines s_1.

2. Upon reception of all such values P_1 finds the processor P_s with the smallest s_i, $1 \leqslant i \leqslant p$.

3. P_1 discards the smallest α elements from P_s's set.

   P_1 discards the largest $(\alpha - 1)$ elements from P_i's set, $1 \leqslant i \leqslant p$, $i \neq s$.

   $k \leftarrow k - \alpha$.

   If $k > 1$, go to step 1; otherwise find the smallest value at each P_i which we call x_i, $1 \leqslant i \leqslant p$. The smallest of all such x_i's is the value sought.

LANSA 3 differs from the original description of **Distributed Sorting** in

[RSS 85] in that phase 1, which is the local sort phase, is an added feature. This modification was done to make distributed selection in the second phase of the algorithm more efficient. SEL 1 and SEL 2 require us to determine the $\lfloor \pi k/n \rfloor^{th}$ value at P_l and the $\lfloor (k-1)/p \rfloor^{th}$ smallest element at each P_i respectively. The original algorithm determines these values using a local selection algorithm at each processor. Since each processor determines $(p-1)(p+\log\log k)$ values on the average, the local processing costs to perform an additional local sort at each processor are dwarfed by the local processing costs to perform the local selections.

Another point worth mentioning is that sometimes the block size (M) at the end of the sort may differ from the original block size. This situation arises when an $F[k]$ is found to be a duplicate; however, our experimental results show that this difference is within 1 % of the original block size.

2.6. LANSA 4

LANSA 4 has been adapted from the **Distributed Quicksort** algorithm in [Weğ 84]. Quicksort is a recursive sorting algorithm that partitions the keys of a file F into two subfiles, F_1 and F_2, such that all keys in F_1 are smaller than a randomly chosen pivot value v, and all keys in F_2 are greater than v. The two subfiles are then sorted independently and recursively using Quicksort.

Consider a file F containing N keys that is distributed among p processors such that a block of $M = \lceil N/p \rceil$ keys is stored in each processor's local memory. F can be partitioned into two subfiles F_1 and F_2 by letting each processor P_i, $1 \leqslant i \leqslant p$, independently partition its block using the same pivot value v. To obtain v, we let P_1 be the controlling processor, and ask each processor P_j, $2 \leqslant j \leqslant p$ to report m_j, the median of its block to P_1. P_1, in the meantime, also determines m_1. This phase is called the **Median phase**. P_1 calculates the mean of the m_i's, $1 \leqslant i \leqslant p$, to obtain v which is then broadcast. Each median, m_i, is determined using the local selection algorithm from [SFR 83]. Let l_i and h_i be the number of keys $\leqslant v$ and $> v$, respectively, in P_i $(1 \leqslant i \leqslant p)$. The process of determining the l_i's and h_i's is called the **Qsort phase**. Since v is the mean of the medians, we assume that the sum of the l_i's gives the global split position for F at processor $P_{p/2}$ (assuming p is even). Keys can now be exchanged between P_1 and $P_{p/2+1}$, P_2 and $P_{p/2+2}$, ... $P_{p/2}$ and P_p to obtain two subfiles F_1 and F_2. This phase is called the **Exchange phase**. F_1 and F_2 are sorted with the same method. The algorithm stops when the subfiles are of length$\approx M$ after which the subfiles can then be sorted locally within a processor, using a quicksort algorithm. Figure 2-5 illustrates this algorithm for four processors, where $M = 5$.

The local k-select algorithm from [SFR 83] has been adapted to our model. We call this algorithm SEL 3. Let S be the set containing the elements. We are interested in finding the k^{th} smallest element of S. The algorithm consists of two steps:

P1	4	11	20	7	8
P2	19	17	3	7	9
P3	2	5	13	16	12
P4	10	1	18	15	14

Qsort phase v[P1. P2. P3. P4] = 10

P1	4	8	7	20	11
P2	9	7	3	17	19
P3	2	5	13	16	12
P4	10	1	18	15	14

Exchange Phase

P1	4	8	7	2	5
P2	9	7	3	10	1
P3	20	11	13	16	12
P4	17	19	18	15	14

Qsort phase v[P1. P2] = 6 v[P3. P4] = 15

P1	4	5	2	7	8
P2	1	3	7	10	9
P3	12	11	13	16	20
P4	14	15	18	19	17

Exchange Phase

P1	4	5	2	1	3
P2	7	8	7	10	9
P3	12	11	13	14	15
P4	16	20	18	19	17

Local sort

P1	1	2	3	4	5
P2	7	7	8	9	10
P3	11	12	13	14	15
P4	16	17	18	19	20

Figure 2-5: LANSA 4

1. Choose an element x at random from S.

2. Partition the elements of S into 3 subsets SL, SE, SG which contain the elements of S which are smaller, equal, and larger than x, respectively. Then

   a. If $k \leqslant |SL|$, $S = SL$
   i.e., discard values $> x$ and return to step 1.

   b. If $|SL| < k \leqslant |SL| + |SE|$, halt: x is the value sought.

   c. If $k > |SL| + |SE|$, $S = SG$

   $k \leftarrow k - |SL| - |SE|$
   i.e., discard values $\leqslant x$ and return to step 1.

LANSA 4 differs from the original description of **Distributed Quicksort** in that the global split position for F is assumed to be at processor $P_{p/2}$ (assuming p is even) while the original algorithm may split the file at any processor P_g, $1 \leqslant g \leqslant p$. This modification can lead to the block size M at the end of the sort being different from the original block size; however, our experimental results show that this difference is within 1 % of the original block size.

Chapter 3

Theoretical Cost Analysis

In this chapter, we perform average case cost analyses of the LANSAs according to our theoretical model. Our theoretical cost model considers both the communication costs and local processing costs to be important quantities in evaluating the efficiency of the LANSAs. The local processing cost is the average number of parallel comparison-exchanges performed by the processors. The number of parallel packet transmissions in the network is a lower bound on the communication cost (maximum communication parallelism) while the total number of packets transmitted in the network is an upper bound (minimum communication parallelism).

Our model is incomplete in the following ways. First, we do not specify what weights should be given to the cost components because we do not know these weights.

Second, it is difficult to estimate the degree of *communication parallelism* (defined below). Also the program and system overheads cannot be estimated and are thus ignored.

Third, our model does not consider packet **collisions** in the network or buffer overflow. When two or more processors attempt to transmit at precisely the same time, a packet collision occurs. A processor recovers from a detected collision by abandoning the attempt and retransmitting the packet. Buffer overflow occurs when several processors attempt to send packets to one processor. This results in a loss of packets and necessitates retransmission of packets. In both cases, additional communication resources are used which are not accounted for in our model.

According to an empirical study on a LAN environment [PaP 85], approximately 90% of the total time for the transmission of a packet is spent on processing of the packet by the transmitting and receiving processors. The network is utilized only 10% of the time. As a result, although a bus type network such as an Ethernet is not sharable, much of the transmission of packets can proceed in parallel. Parallel packet transmission is the transmission of packets that are initiated simultaneously by the processors in the network. Maximum communication parallelism is achieved when the time required to transmit two or more packets is (almost) the same as the time required to transmit one packet. In contrast, minimum communication parallelism is the serial transmission of packets.

Before we analyze the algorithms, we need some definitions. Let F be the global file containing N keys and let p be the number of processors available to perform the sort. The keys are distributed equally among the p processors so that a block of $M = \lceil N/p \rceil$ keys is stored in each processor's local memory. Let S be the maximum size of a packet (in number of keys) that can be transmitted through the network.

3.1. LANSA 0

In this algorithm, processor P_1 receives blocks of data from processors $P_2, P_3, \ldots, P_p$. P_1 performs a local sort using a quicksort algorithm and retransmits the sorted blocks back to these processors. The local sort takes

$$c_0 N \log N \text{ comparisons}$$

on average, where c_0 is a constant.

The lower bound for the communication cost is

$$2\lceil M/S \rceil \text{ packets}$$

while the upper bound is

$$2(p-1)\lceil M/S \rceil \text{ packets.}$$

3.2. LANSA 1

Recall that this algorithm consists of $(p+1)$ steps. Step 0 is a local sort which is performed independently by each processor. During the odd (even) steps, the odd- (even-) numbered processors receive from their higher-even (higher-odd) numbered neighbor a sorted block, perform a two-way merge, and send back the higher M keys. Thus, the local processing cost consists of the local sort and p merge-split steps. The local sort takes $c_1 M \log M$ comparisons on average and a merge-split takes $2M$ comparisons. Hence, the total local processing cost is

$$c_1 M \log M + 2M \cdot p \text{ comparisons}$$

on average, where c_1 is a constant.

The lower bound for the communication cost per step is $2\lceil M/S \rceil$ packets. Thus, p steps require

$$2p\lceil M/S \rceil \text{ packets}$$

to be transmitted in the network. During an odd (even) step, p $(p-2)$ processors transmit $p\lceil M/S \rceil$ $((p-2)\lceil M/S \rceil)$ packets in total. Since there are $p/2$ odd steps and $p/2$ even steps, the upper bound for the communication cost evaluates to

$$p(p-1)\lceil M/S \rceil \text{ packets.}$$

3.3. LANSA 2

This algorithm consists of $(\log p + 2)$ stages. Stage 0 consists of each processor locally sorting two blocks of data of size $M/2$. Stage 1 consists of a merge-split step performed by each processor. Stage i, $2 \leqslant i \leqslant \log p$, consists of $(i-1)$ shuffles, i merge-splits and a transmission step. Finally, stage $(\log p + 1)$ consists of $(\log p + 1)$ shuffles and $(\log p + 1)$ merge-splits. To sort a block of

$M/2$ keys requires $c_2 M(\log M - 1)/2$ comparisons on average and to merge two blocks of $M/2$ keys requires M comparisons. Hence, the total local processing costs are

$$c_2 M(\log M - 1) + (\log p + 1)\,(\log p + 2)M/2 \text{ comparisons}$$

on average, where c_2 is a constant.

The communication costs are incurred during stage i $(2 \leqslant i \leqslant \log p)$ and stage $(\log p + 1)$. Stage i consists of $(i - 1)$ shuffles and one transmission step; stage $(\log p + 1)$ consists of $(\log p + 1)$ shuffles. Although a shuffle step is different from a transmission step, the number of packets transmitted is the same. We can, therefore, assume that stage i $(2 \leqslant i \leqslant \log p + 1)$ consists of i transmission steps. This gives us a total of $(\log p)\,(\log p + 3)/2$ transmission steps required for the entire sort. Thus, the lower bound for the communication cost is

$$(\log p)\,(\log p + 3)\,\lceil M/2S \rceil/2 \text{ packets.}$$

The upper bound for the communication cost must take into account the number of actual output lines in a shuffle or a transmission step. Since there are $2(p - 1)$ actual output lines in a shuffle or a transmission step, the upper bound for the communication costs is

$$(\log p)\,(\log p + 3)\,(p - 1)\lceil M/2S \rceil \text{ packets.}$$

3.4. LANSA 3

In LANSA 3, the local processing costs are incurred during the two local sorts, the selection phase, and the assignment phase. The two local sorts use $2c_1 M \log M$ comparisons on average and the assignment phase requires M comparisons.

To determine the average complexity of the selection phase, we have to make some assumptions. Note that the k^{th} smallest element is determined for $k = M, 2M, \ldots, (p-1)M$. During this phase, P_1 broadcasts x and requests, in return, the number of elements $\leqslant x$. Let us assume that every time this request is made by P_1, each processor (including P_1) searches through its file and finds an average of k/p elements $\leqslant x$. This takes k/p comparisons. The assumption is valid since our keys are generated randomly and are uniformly distributed. In [SaS 83], it is proved that the algorithm iterates log log k times, on the average, before it finds the k^{th} smallest element. Thus, it takes $(\log \log k)k/p$ comparisons to find the k^{th} smallest element. Since the k^{th} smallest element is determined for $k = M, 2M, \ldots, (p-1)M$, the selection phase takes an average of $(\log \log M)M/p + (\log \log 2M)2M/p + \cdots + (\log \log (p-1)M)\,(p-1)M/p$ comparisons at each processor. Hence, the total local processing costs are

$$M(2c_1 \log M + 1 + 1/p \sum_{i=1}^{p-1} \log \log (i \cdot M)\, i\,) \text{ comparisons}$$

on average, where c_1 is a constant.

Communication costs, in this algorithm, are due to the selection and routing phases. In [SaS 83], the authors analyze the distributed selection algorithm in detail and derive an upper bound on its average complexity. They show that the algorithm requires $(p + \log \log k)$ basic communication activities, on the average, to determine the k^{th} smallest element. A **basic communication activity** (bca) consists of a processor broadcasting a message and receiving a reply from all other processors. In our model, the lower bound for a bca is 2 packets while the upper bound is $2(p-1)$ packets. Since the k^{th} smallest element is determined for $k = M, 2M, \ldots, (p-1)M$, the lower bound for the selection phase is $2(p + \log \log M + p + \log \log 2M + \cdots + p + \log \log (p-1)M)$ packets on average. This simplifies to $2(p(p-1) + \Sigma_{i=1}^{p-1} \log \log (i \cdot M))$ packets. The upper bound for the selection phase is $2(p-1)\,(p + \log \log M + p + \log \log 2M + \cdots + p + \log \log (p-1)M)$ packets on average. This simplifies to $2(p-1)\,(p(p-1) + \Sigma_{i=1}^{p-1} \log \log (i \cdot M))$ packets.

In the routing phase, each key is sent to its final destination. To analyze this phase, assume that $\lceil M/p \rceil$ keys (on the average) are transmitted by each processor P_i to each processor P_j $(1 \leqslant i,j \leqslant p;\, i \neq j)$. The lower bound for the routing phase is, thus, $p\lceil M/Sp \rceil$ packets on average while the upper bound is $p(p-1)\lceil M/Sp \rceil$ packets on average. Hence, the lower bound for the communication cost is

$$2p(p-1) + p\lceil M/Sp \rceil + 2 \sum_{i=1}^{p-1} \log \log (i \cdot M) \text{ packets}$$

on average while the upper bound is

$$(p-1)\,(2p(p-1)+p\lfloor M/Sp\rfloor+2\sum_{i=1}^{p-1}\log\log\,(i\cdot M)\,)\text{ packets}$$

on average.

3.5. LANSA 4

The local processing costs are incurred during the median phase, the qsort phase, and the local sort phase. The median and the qsort phases are executed (log p) times while the local sort is performed at the end. To determine the median, we used the sequential selection algorithm by [SFR 83]. [SFR 83]'s algorithm, on average, requires c_3M comparisons to determine the median. The qsort phase is similar to the "partioning/swapping" operation in quicksort and requires M comparisons. Finally, the local sort requires $c_1M\log M$ comparisons on average. Hence, the total local processing costs are

$$(M\log\,p)\,(c_3+1)+c_1M\log\,M\text{ comparisons}$$

on average, where c_1 and c_3 are constants.

The communication costs are incurred when the pivot value v is being determined and during the exchange phase. The pivot value v is determined as follows. Each processor P_j $(2\leqslant j\leqslant p)$, after determining its median, transmits it to P_1. P_1 calculates v and broadcasts it to P_j. This process is performed (log p) times. Thus, the lower bound for determining v and broadcasting it is 2(log p) packets. The upper bound is $2(p-1)$ (log p) packets.

In the exchange phase, each processor exchanges keys with one other processor to obtain the two subfiles F_1 and F_2. To analyze this phase, assume that after the qsort phase is over, each processor has to transmit $M/2$ keys (on the average) to obtain the subfiles F_1 and F_2. Since this phase is done ($\log p$) times, the lower bound for this phase is $(\log p)\lceil M/2S\rceil$ packets while the upper bound is $(\log p)p\lceil M/2S\rceil$. Hence, the lower bound for the communication cost is

$$(\log p)\ (\lceil M/2S\rceil + 2) \text{ packets}$$

on average while the upper bound is

$$(\log p)\ (2(p-1) + p\lceil M/2S\rceil) \text{ packets}$$

on average.

3.6. Summary

In this section, we summarize the theoretical results derived in this chapter. Table 3-1 shows the average local processing costs (in number of comparisons) while table 3-2 shows the lower and upper bounds for the communication costs (in average numbers of packets).

Algorithm	Local Processing Costs (comparisons)
LANSA 0	$c_0 N \log N$
LANSA 1	$c_1 M \log M + 2M \cdot p$
LANSA 2	$c_2 M(\log M - 1) + (\log p + 1)\ (\log p + 2)M/2$
LANSA 3	$M(2c_1 \log M + 1 + 1/p\ \Sigma_{i=1}^{p-1} \log\log (i \cdot M)\ i\ )$
LANSA 4	$(M \log p)\ (c_3 + 1) + c_1 M \log M$

Table 3-1: Local Processing Costs

Communication Costs (packets)

Algorithm	**Lower Bound**	**Upper Bound**
LANSA 0	$2\lceil M/S \rceil$	$2(p-1)\lceil M/S \rceil$
LANSA 1	$2p\lceil M/S \rceil$	$p(p-1)\lceil M/S \rceil$
LANSA 2	$(\log p)\,(\log p+3)\,\lceil M/2S \rceil/2$	$(\log p)\,(\log p+3)\,(p-1)\lceil M/2S \rceil$
LANSA 3	$2p(p-1)+p\lceil M/Sp \rceil +$ $2\,\Sigma_{i=1}^{p-1} \log\log\,(i{\cdot}M)$	$(p-1)\,(2p(p-1)+p\lceil M/Sp \rceil +$ $2\,\Sigma_{i=1}^{p-1} \log\log\,(i{\cdot}M)\,)$
LANSA 4	$(\log p)\,(\lceil M/2S \rceil+2)$	$(\log p)\,(2(p-1)+p\lceil M/2S \rceil)$

Table 3-2: Communication Costs

From the above summaries, it can be seen that for each LANSA

- the local processing costs <u>decrease</u> as the number of processors to perform the sort <u>increases</u>.
- the communication costs <u>increase</u> as the number of processors to perform the sort <u>increases</u>.

We cannot make an overall relative comparison of these algorithms because

- the constants are unknown, and
- we have not attached any weights to the local processing and communication cost components.

Chapter 4

Experimental Results and Analysis

In this Chapter we first describe the experiment. In the rest of the Chapter, we present and analyze the experimental results for the five LANSAs described in Chapter 2.

4.1. Experiment

4.1.1. Hardware and Software Requirements

The experiments were performed on a network of 18 Sun workstations connected by an Ethernet, a high-speed LAN. Ethernet is a popular local area broadcasting network used for local communication among computing stations. The 10 megabits per second Ethernet can effectively handle data traffic for hundreds of stations within an area of 2 square kilometres.

Our facility is a homogeneous system. Each Sun workstation (Model Sun-2) uses the 10-MHz version of Motorola's 32-bit 68010 microprocessor and comes with 2 MB of main memory. These workstations run the same operating system, Sun UNIX, and have access to the same file system. The operating system supplies inter-network communication primitives and services allowing users to make use of the distributed nature of the facility. In our experiments, we used the **Transmission Control Protocol** (TCP) for transmission of packets over Ethernet. TCP is a reliable transmission protocol that limits the maximum Ethernet packet size to 1024 bytes (256 keys).

Of the 18 Sun workstations, three are file servers. Each file server has 2 or 4 MB of main memory and a 380 MB disk drive with controller. The server systems provide the remaining diskless workstations with shared disk storage for file systems and paging, and shared access to other peripherals. During the experiments, we avoided using the file servers to perform the sort for obvious reasons.

The facility provided us with all the hardware and software required to carry out the experiments. The experiments were performed on four and eight processors using various global file sizes. In each case, we used two equal size sets of diskless workstations. In addition, two file servers were used -- one for each set of workstations.

4.1.2. Data Collection

The LANSAs were run on four and eight processors with the configurations mentioned in section 4.1.1. All of our experiments were run on a dedicated system so that timing measurements would be accurate.

In our experiments, the files consisted of keys which were integers. The keys were generated randomly and were uniformly distributed over a given range. Each processor P_i, $1 \leqslant i \leqslant p$, generated M keys (block size). Duplicate keys were possible both locally (within a block) and globally (over the entire file F). The LANSAs performed the sort successfully on global file sizes of $2K$, $4K$, $8K$ and $16K$ keys (where $K = 1024$).

All our LANSAs consist of several phases or steps. For each phase, the communication costs and local processing costs were measured in terms of *elapsed time* in milliseconds (ms) to give us communication time and local processing time. The sum of these components over all phases yields the *response time* for the entire sort. Note that some phases may only consist of one of the above cost measurement components. The timer that we used was rounded-off to the nearest 10 ms. The results gathered were the average response times over twenty runs. For each run, the same keys were regenerated and at the end of the run the timings were recorded. At the end of all of the runs, the average timings were calculated and stored in a file. Note that the *connection* time for the processors was not included as part of the response time. This connection time, which typically runs in hundreds of milliseconds, is required by the TCP/IP protocol to link the processors before any packets are exchanged. After the connections are made, a selected processor broadcasts a *sync* message to synchronize the processors. The sync message is four bytes long and is broadcast at the beginning of each run after which timing measurement begins in each processor.

4.2. Methodology of the Analysis

The rest of this chapter is devoted to the analysis of the five LANSAs described in Chapter 2. The algorithms are analyzed in the following manner. There are two parameters that affect the performance of the algorithms: N and p. During the experiment, various files of different sizes were created and run on four and eight processors. In section 4.3, we describe the behaviour of the algorithms based on the parameter N. Section 4.4 ranks the algorithms based on parameter p and assuming N is large ($16K$ keys). In section 4.5, we present the analysis of each algorithm as parameter p increases leaving N constant at $16K$ keys.

4.3. Size of the Data Sets

Figure 4-1 and figure 4-2 show a plot of the performance of the five LANSAs for $p=4$ and $p=8$ respectively. The results are also summarized in table B-1 and table B-2 in Appendix B. Table B-3 and table B-4 in Appendix B show the breakdown of the response time in terms of local processing cost and communication cost.

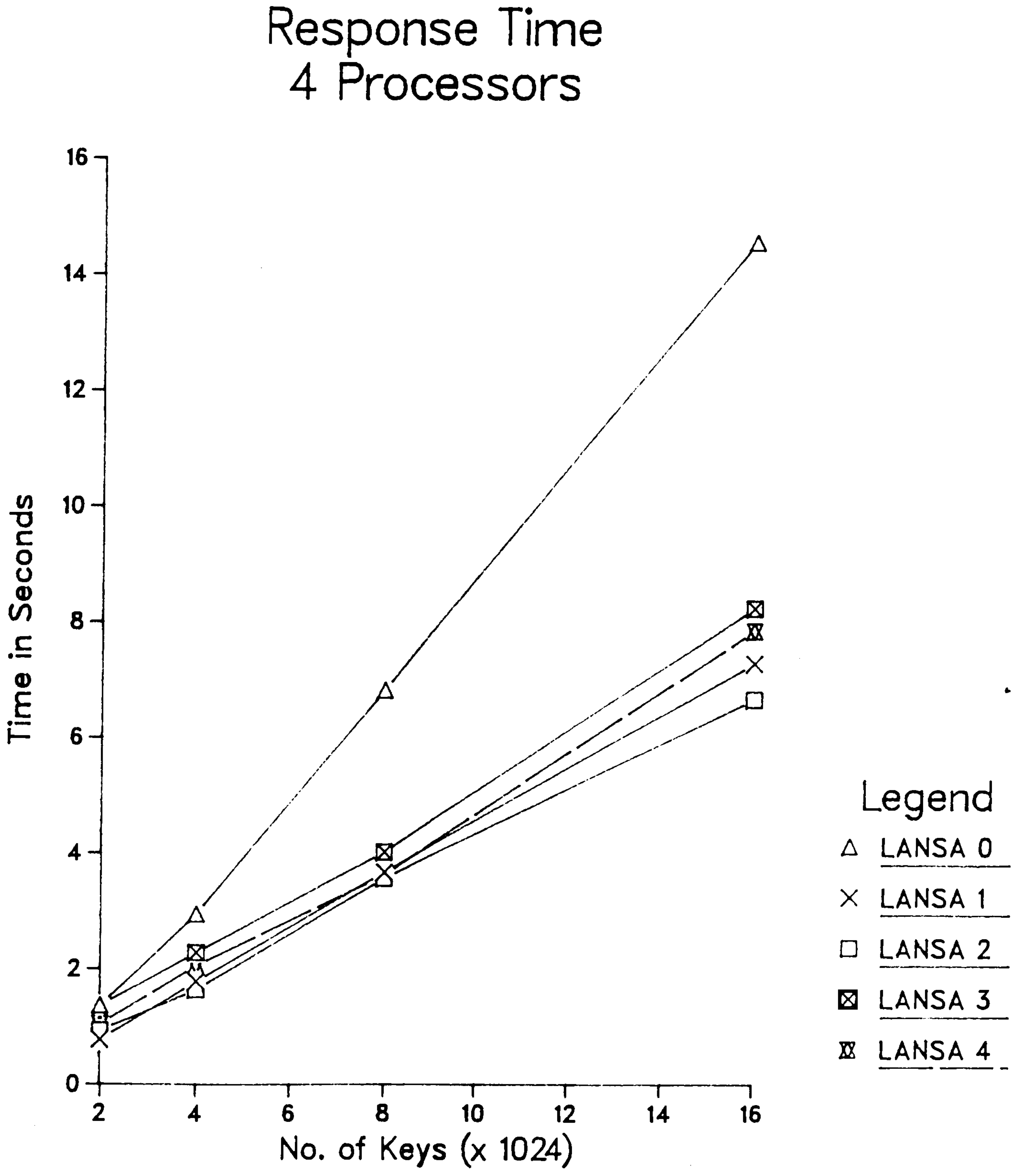

Figure 4-1: Response Time for $p=4$

Normally, the performance of a sorting algorithm is dependent on the size of the data set i.e. the larger the data set, the longer the algorithm takes to sort the keys. Figure 4-1 and figure 4-2 show that all the LANSAs behave normally except LANSA 3 for the case $p=8$.

4.3.1. LANSA 3

The behaviour of this algorithm is quite erratic for $p=8$. The interesting situation is that the algorithm does better when sorting $4K$ keys than when sorting $2K$ keys(see figure 4-2).

The graph in figure 4-3 shows the response time, local processing cost component and communication cost component for LANSA 3. It can be seen that local processing cost increases as N increases, so the interesting behaviour is in the communication component of the algorithm. The communication cost is incurred in the selection phase and the routing phase. Table 4-1 shows the communication costs for the selection and the routing phases.

From table 4-1, it is clear that the selection phase is responsible for the peculiar behaviour of this algorithm. In fact, the communication cost in the selection phase is higher at $N=2K$ than at any other value of N. This suggests that the selection algorithms go through more iterations to find the k^{th} smallest element for $N=2K$ than for any other value of N. Recall that we used SEL 1

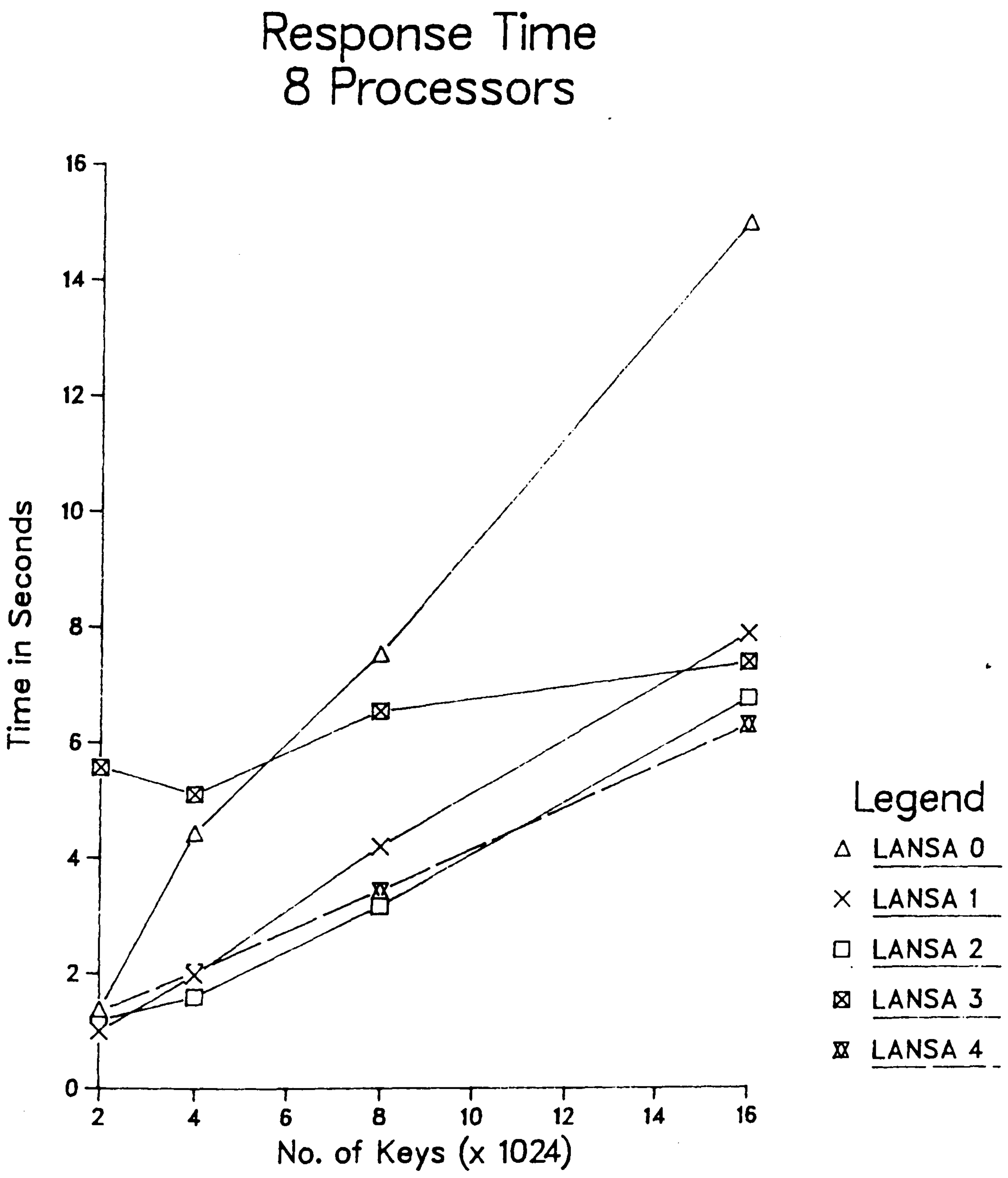

Figure 4-2: Response Time for $p = 8$

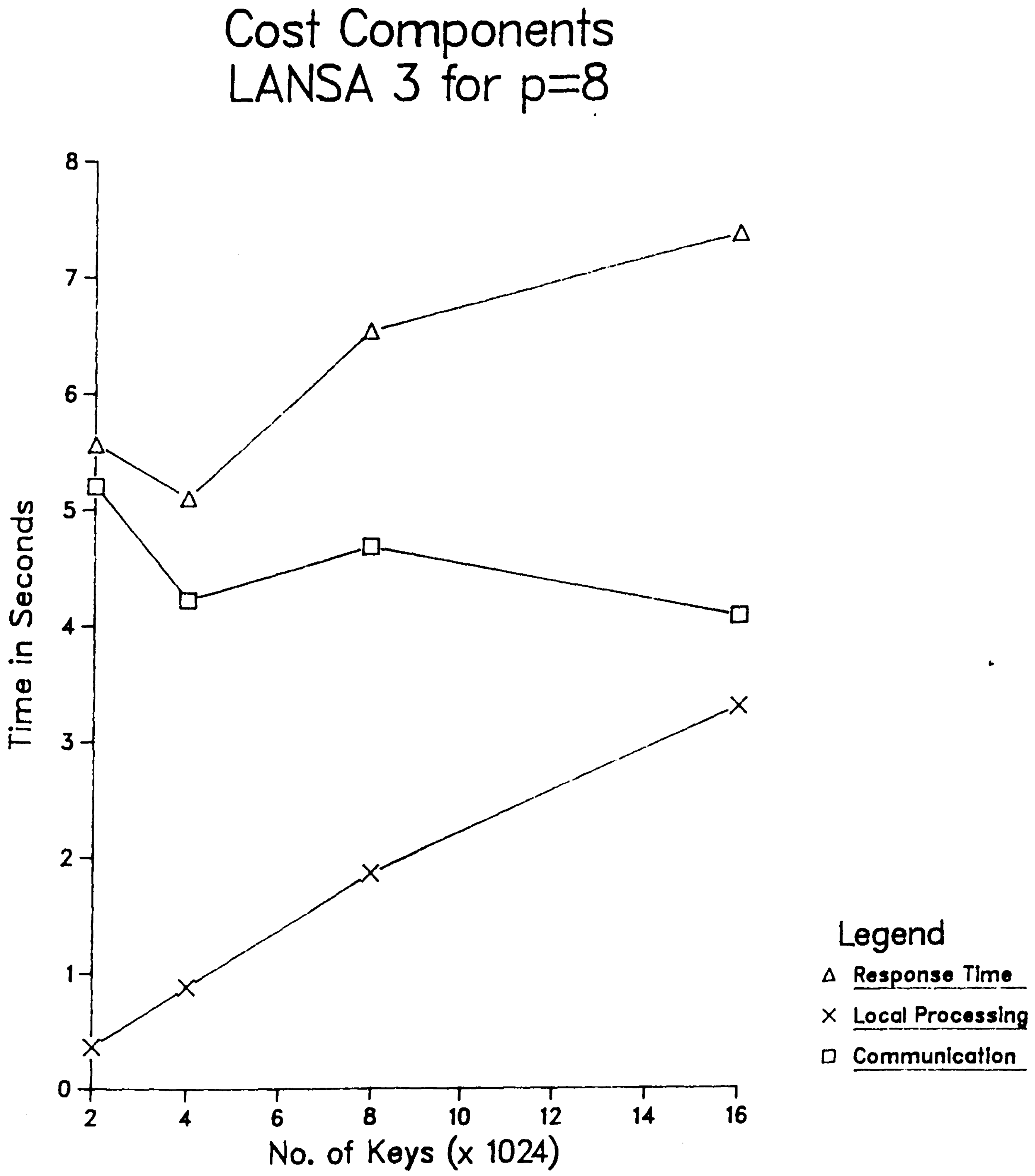

Figure 4-3: Cost Components

	N			
Phase	**2*K***	**4*K***	**8*K***	**16*K***
Selection	4.98	3.96	4.36	3.52
Routing	0.22	0.26	0.32	0.56

Table 4-1: Communication Costs in Seconds

and SEL 2 to determine the k^{th} smallest element. SEL 1 is a reduction algorithm that goes through a sequence of iterations to reduce the size of the problem to a value smaller than T (termination criterion). When the termination criterion is met, SEL 2 is employed to determine the k^{th} smallest element. We chose T experimentally. The number of iterations SEL 1 and SEL 2 went through were recorded for all values of N and p. A single value of T was chosen for $p=4$ and was used for all values of N. Similarly, a single value of T was chosen for $p=8$ and was used for all values of N. One possible reason why SEL 1 and SEL 2 behave in this manner could be that the value of T we chose was not suitable for all values of N. We think that more comprehensive tests on T for each N may provide us with more insight into the nature of LANSA 3.

4.4. Ranking of the Algorithms

4.4.1. General Ranking

In this section, we rank the algorithms according to their performance for $N=16K$. Table 4-2 shows the algorithms ranked for $p=4$ while table 4-3 shows the ranking for $p=8$.

Rank	Algorithm	Response Time
1	LANSA 2	6667
2	LANSA 1	7287
3	LANSA 4	7852
4	LANSA 3	8240
5	LANSA 0	14546

Table 4-2: Algorithm Ranks for $p=4$ and $N=16K$

Rank	Algorithm	Response Time
1	LANSA 4	6294
2	LANSA 2	6758
3	LANSA 3	7380
4	LANSA 1	7873
5	LANSA 0	14984

Table 4-3: Algorithm Ranks for $p=8$ and $N=16K$

The experimental results show that LANSA 2 is the best algorithm for $p=4$, while LANSA 4 is the best algorithm for $p=8$. LANSA 3 and LANSA 4 (distributed algorithms) perform better when run on eight processors than when run on four processors. LANSA 1 and LANSA 2 (parallel algorithms) perform better when run on four processors than when run on eight processors. The next subsection discusses the change of ranking from $p=4$ to $p=8$.

4.4.2. Local Processing vs Communication Costs

Table 4-4 and table 4-5 show the local processing costs and communication costs for $p=4$ and $p=8$ respectively.

	Cost Component	
Algorithm	Local Processing Cost	Communication Cost
LANSA 0	13450 (92%)	1096 (8%)
LANSA 1	4252 (58%)	3035 (42%)
LANSA 2	4386 (66%)	2281 (34%)
LANSA 3	6800 (83%)	1440 (17%)
LANSA 4	5050 (64%)	2802 (36%)

Table 4-4: Cost Components for $p=4$ and $N=16K$

	Cost Component	
Algorithm	Local Processing Cost	Communication Cost
LANSA 0	13143 (87%)	1841 (13%)
LANSA 1	2584 (33%)	5289 (67%)
LANSA 2	2487 (37%)	4271 (63%)
LANSA 3	3300 (56%)	4080 (44%)
LANSA 4	3310 (53%)	2984 (47%)

Table 4-5: Cost Components for $p=8$ and $N=16K$

It can be seen that the local processing cost is lower for $p=8$ than for $p=4$. The reason for this is that the size of the local file (i.e., block size M) at each processor is smaller in the case of $p=8$ than in the case of $p=4$, so local processing activities such as local sorts take less time.

The communication cost is higher for $p=8$ than for $p=4$. One reason for this increase may be that more processors are contending for access to the Ethernet. This should result in more packets collisions. Packets that collide have to be retransmitted, and this requires additional communication resources.

In our theoretical model, we assumed that both local processing costs and communication costs are important. Table 4-4 and 4-5 show that, indeed, communication costs and local processing costs are both important quantities and neither can be ignored.

It is interesting to compare the cost components of the algorithms with their design principles. LANSA 1 and LANSA 2 (parallel algorithms) were designed to minimize local processing cost while LANSA 3 and LANSA 4 (distributed algorithms) were designed to minimize communication cost. These principles are reflected in tables 4-4 and 4-5. However, it should be noted that LANSA 3 has a large increase in communication cost from $p=4$ to $p=8$. The fact that its total response time is reduced from $p=4$ to $p=8$ is due to the drastic reduction in processing cost.

4.4.3. Improvement of the Network Performance

In this section, we extrapolate the behaviour of the LANSAs assuming that the performance of the network can be improved. This improvement of the network concerns the interprocess communication (IPC) facilities built into Sun UNIX. The basic building block for communication in Sun UNIX is the *socket*. A socket is an endpoint of communication. These sockets can be connected to form bidirectional communication streams. However, since they were designed to be a general-purpose robust communication mechanism, the overhead in using the socket facility is substantial. If we were to use an IPC design which is performance oriented such as the *V Kernel* [Che 84], the communication costs would decrease and this would improve the overall performance of the LANSAs.

We are doing this extrapolation because two of the algorithms being considered here were designed for an environment where communication costs were low or negligible. We would like to compare these algorithms and predict the behaviour when the communication facility is much improved with respect to the local processing facilities. Table 4-6 and table 4-7 show the extrapolation for $p=4$ and $p=8$ respectively. The improved response time (in seconds) is obtained by adding the known local processing costs to the diminished communication costs.

Table 4-6 shows that improving the network performance by 90% when 4 processors are used will only affect the relative performances of LANSA 1 and LANSA 2. The outcome is that LANSA 1 will perform better than LANSA 2.

Algorithm	Proc. Costs	Comm. Costs	Response Time	% Improvement of Network Performance 30%	60%	90%
LANSA 0	13.450	1.096	14.546	14.217	13.888	13.56
LANSA 1	4.252	3.035	7.287	6.377	5.466	4.556
LANSA 2	4.386	2.281	6.667	5.983	5.298	4.615
LANSA 3	6.800	1.440	8.240	7.808	7.376	6.944
LANSA 4	5.050	2.802	7.852	7.011	6.171	5.330

Table 4-6: Improved Response Time (in seconds) for $p=4$

Algorithm	Proc. Costs	Comm. Costs	Response Time	% Improvement of Network Performance 30%	60%	90%
LANSA 0	13.143	1.841	14.984	14.428	13.877	13.23
LANSA 1	2.584	5.289	7.873	6.286	4.700	3.113
LANSA 2	2.487	4.271	6.758	5.477	4.195	2.914
LANSA 3	3.300	4.080	7.380	6.156	4.932	3.708
LANSA 4	3.310	2.984	6.294	5.399	4.504	3.608

Table 4-7: Improved Response Time (in seconds) for $p=8$

Table 4-7 shows that the improvement of the network can significantly affect the relative performances of the LANSAs when the number of processors is increased. If the network performance is improved by 60%, LANSA 2 trades places with LANSA 4 and LANSA 1 trades places with LANSA 3. A 90% improvement results in LANSA 1 performing better than LANSA 4. However, in both the cases (60% and 90% improvement), LANSA 2 performs better than all the other LANSAs.

4.5. Four Processors vs Eight Processors

In this section, the performance of the algorithms for $p=4$ is compared to the performance of the algorithms for $p=8$, with constant $N(=16K)$. We analyze each algorithm, and compare the expected performance with the actual (experimental) performance. we describe why our results are not as anticipated. We are particularly interested in the amount of the communication parallelism (CP) that can be achieved by the different algorithms and the way the communication parallelism changes as the number of processors increases.

Experimentally, communication parallelism can occur between the point a transmitting processor starts to send a message and the point a receiving processor receives this message. Between these extreme points, a message goes through a packet building and buffer management process (at the transmitting end), packet transmission, and a buffer management and packet disassembly process (at the receiving end). During the actual packet transmission process, no parallelism can be attained in a broadcast network such as an Ethernet. However, a very small portion of the time between the two extreme points is devoted to the actual transmission of the packet. Thus, parallelism can be achieved during the packet building and buffer management processes among contending processors, and a user gets the illusion that packets can be transmitted through the network in parallel.

A LANSA achieves maximum CP if the time required to transmit all simultaneously initiated messages is equal to the time required to transmit a single message. In contrast, a LANSA achieves minimum CP when the time required to transmit all simultaneously initiated messages is equal to the time required to transmit these messages serially. The lower and upper bounds derived in Chapter 3 are theoretical bounds on the CP possible for each LANSA. If the maximum degree of CP is assigned a value of 1 and the minimum degree of CP is assigned a value of 0, then the degree of CP for a LANSA is c, where $0 \leqslant c \leqslant 1$. In practice, the transmission of simultaneously initiated messages cannot be entirely parallel and is usually not entirely serial.

Before investigating the CP of the algorithms, we compare two ratios for each algorithm. *R1* is the ratio of the theoretical local processing costs for $p=8$ and $p=4$, and *R2* is the ratio of the experimental local processing costs for $p=8$ and $p=4$. We expect that $R2 \approx R1$, i.e., the experimental results agree with the theoretical results. If $R2 \approx R1$, then we can assume that our timing function used in the experiments is accurate.

The theoretical upper and lower bounds on communication costs derived in Chapter 3 were expressed in terms of number of packets transmitted. Our experimental results, however, were measured in terms of elapsed time in milliseconds. Since the theoretical bounds are in different units than the experimental results, we cannot directly compare them. Nonetheless, we can indirectly compare them as follows.

Let t be the time in milliseconds needed to transmit one packet (including the packet building and disassembly times and the buffer management times at both sending and receiving ends) when no packet collisions or buffer overflows occur. Experimentally, t was found to be 16 milliseconds. For each particular LANSA, let t_4 and t_8 be the total empirically measured time (in milliseconds) during which packets were transmitted for $p=4$ and $p=8$ respectively. Then $a_4=t_4/t$ and $a_8=t_8/t$ are the numbers of "intervals" during which packets were transmitted. In other words, a_4 and a_8 are the experimental numbers of parallel packet transmissions. a_4 and a_8 are in the same units as the theoretically determined lower and upper bounds. Let u_4 (l_4) and u_8 (l_8) be the theoretical upper (lower) bounds for $p=4$ and $p=8$ respectively.

Definition 1: The degree of communication parallelism for an algorithm is

$$c_p = \frac{u_p - a_p}{u_p - l_p}$$

where $p=4$ or $p=8$ is the number of processors.

If no packet collisions or buffer overflows occur, then $l_p \leqslant a_p \leqslant u_p$ and it follows directly that $0 \leqslant c_p \leqslant 1$. As a_p approaches l_p, c_p approaches 1 (maximum CP) and as a_p approaches u_p, c_p approaches 0 (minimum CP). Although it is rarely the case that no packet collisions or buffer overflows occur, c_p should give us at least a general idea of how much communication parallelism the LANSAs exhibit.

4.5.1. LANSA 0

The local processing cost is $c_0 N \log N$ comparisons. For $p=4$ or $p=8$, this evaluates to the same number of comparisons. Consequently, $R1=1$. The experimental ratio $R2$ evaluates to

$$\frac{13143}{13450} \approx 1$$

Thus, the theoretical ratio $R1$ is approximately equal to the experimental ratio $R2$ as expected.

The lower bound for the communication cost is $2\lceil M/S \rceil$ packets while the upper bound is $2(p-1)\lceil M/S \rceil$ packets. We evaluate these expressions for $p=4$, $M=4K$ and $p=8$, $M=2K$ to give us l_4, u_4 and l_8, u_8 respectively. The results are summarized in table 4-8.

	Number of Packets	
Bound	$p=4$	$p=8$
Lower	32	16
Upper	96	112

Table 4-8: LANSA 0 - Communication Costs

Using the definition of c_p, $c_4=0.43$ and $c_8=-0.03$. c_4 suggests that there was some communication parallelism when $p=4$ while c_8 suggests that there was no communication parallelism when $p=8$. We expected the amount of communication parallelism to be small in both cases. In LANSA 0, processor P_1

receives blocks of data from $P_2, P_3, \ldots, P_p$. Since these processors all try to send their data at the same time, packets are bound to collide. Furthermore, P_1 receives data from only one processor at a time; thus data from other processors is ignored. This all leads to retransmission of packets. After P_1 has sorted the data, it transmits blocks back to $P_2, P_3, \ldots, P_p$. Since P_1 is the only transmitting processor, no parallelism is possible in this part of the algorithm. One possible reason that c_4 is higher than c_8 is that fewer processors are contending for use of the network, so fewer collisions occur during the first phase when P_1 is receiving blocks of data.

4.5.2. LANSA 1

The local processing cost is $c_1 M \log M + 2M \cdot p$ comparisons. If we evaluate this expression for $p=4, M=4K$ we get $2^{12}(12c_1 + 2^3)$ comparisons while for $p=8, M=2K$ we get $2^{11}(11c_1 + 2^4)$ comparisons. Thus, the theoretical ratio *R1* is

$$\frac{11c_1 + 16}{24c_1 + 16}$$

If $c_1 = 0$, then $R1 \approx 1$. If c_1 is large, then $R1 \approx 0.5$. The experimental ratio *R2* is

$$\frac{2584}{4252} \approx 0.6$$

which is consistent with the bounds on *R1*.

The lower bound for the communication cost is $2p\lceil M/S \rceil$ packets while the upper bound is $p(p-1)\lceil M/S \rceil$ packets. Table 4-9 shows these expressions evaluated for $p=4, M=4K$ and $p=8, M=2K$ to give us l_4, u_4 and l_8, u_8 respectively.

	Number of Packets	
Bound	$p=4$	$p=8$
Lower	128	128
Upper	192	448

Table 4-9: LANSA 1 - Communication Costs

Using the definition of c_p, $c_4=0.04$ and $c_8=0.37$. c_4 exhibits minimal communication parallelism while c_8 shows that there was some communication parallelism. We expected the communication parallelism to be higher than this since processors communicate in pairs at the end of each phase. This should have resulted in minimum packet collisions.

4.5.3. LANSA 2

The local processing cost is $c_2 M(\log M - 1) + (\log p + 1)\ (\log p + 2)M/2$ comparisons. If we evaluate this expression for $p=4, M=4K$ we get $2^{12}(11c_1+6)$ comparisons while for $p=8, M=2K$ we get $2^{11}(10c_1+10)$ comparisons. Thus, the theoretical ratio *R1* is

$$\frac{10c_2+10}{22c_2+12}$$

If $c_2 = 0$, then $R1 \approx 0.8$. If c_2 is large, then $R1 \approx 0.5$. The experimental ratio $R2$ is

$$\frac{2487}{4386} \approx 0.6$$

which is consistent with the bounds on $R1$.

The lower bound for the communication cost is $(\log p)\,(\log p + 3)\,\lceil M/2S \rceil/2$ packets while the upper bound is $(\log p)\,(\log p + 3)\,(p-1)\lceil M/2S \rceil$ packets. Table 4-10 shows these expressions evaluated for $p=4, M=4K$ and $p=8, M=2K$ to give us l_4, u_4 and l_8, u_8 respectively.

	Number of Packets	
Bound	$p=4$	$p=8$
Lower	40	36
Upper	240	496

Table 4-10: LANSA 2 - Communication Costs

Using the definition of c_p, $c_4 = 0.49$ and $c_8 = 0.50$. Both c_4 and c_8 suggest that some communication parallelism has occurred.

4.5.4. LANSA 3

The local processing cost is $M(2c_1 \log M + 1 + 1/p\, \Sigma_{i=1}^{p-1} \log\log (i \cdot M)\, i)$ comparisons. If we evaluate this expression for $p=4, M=4K$ we get $2^{10}(96c_1+26)$ comparisons while for $p=8, M=2K$ we get $2^8(176c_1+112)$ comparisons. Thus, the theoretical ratio *R1* is

$$\frac{176c_1+112}{384c_1+104}$$

If $c_1=0$, then $R1 \approx 1.1$. If c_1 is large, then $R1 \approx 0.5$. The experimental ratio *R2* is

$$\frac{3300}{6800} \approx 0.5$$

which is consistent with the bounds on *R1*.

The lower bound for the communication cost is $2p(p-1) + p\lceil M/Sp \rceil + 2\, \Sigma_{i=1}^{p-1} \log\log (i \cdot M)$ packets while the upper bound is $(p-1)\,(2p(p-1) + p\lceil M/Sp \rceil + 2\, \Sigma_{i=1}^{p-1} \log\log (i \cdot M)\,)$ packets. Table 4-11 shows these expressions evaluated for $p=4, M=4K$ and $p=8, M=2K$ to give us l_4, u_4 and l_8, u_8 respectively.

	Number of Packets	
Bound	$p=4$	$p=8$
Lower	62	171
Upper	186	1200

Table 4-11: LANSA 3 - Communication Costs

Using the definition of c_p, $c_4 = 0.78$ and $c_8 = 0.92$. This suggests that this algorithm achieves high degree of communication parallelism. These figures for communication parallelism were a bit surprising. We expected c_p to be small because of the nature of the selection phase. Recall that most of the communication costs are incurred in the selection phase where processors $P_2, P_3, \ldots, P_p$ all communicate with P_1. Since $P_2, P_3, \ldots, P_p$ try to communicate at the same time, we expected serious buffer overflow problems at P_1. However, it seems that this was not the case here.

4.5.5. LANSA 4

The local processing cost is $(M \log p)(c_3 + 1) + c_1 M \log M$ comparisons. If this expression is evaluated for $p = 4, M = 4K$ we get $2^{12}(12c_1 + 2c_3 + 2)$ comparisons while for $p = 8, M = 2K$ we get $2^{10}(22c_1 + 6c_3 + 6)$ comparisons. Thus, the theoretical ratio *R1* is

$$\frac{22c_1 + 6c_3 + 6}{48c_1 + 8c_3 + 2}$$

If $c_1 = c_3 = 0$, then $R1 \approx 3$. If both c_1 and c_3 are equally large, then $R1 \approx 0.5$. The experimental ratio *R2* is

$$\frac{3310}{5050} \approx 0.7$$

which is consistent with the bounds on *R1*.

The lower bound for the communication cost is $(\log p)(\lceil M/2S \rceil + 2)$ packets while the upper bound is $(\log p)\,(2(p-1)+p\lceil M/2S\rceil)$ packets. Table 4-12 shows these expressions evaluated for $p=4, M=4K$ and $p=8, M=2K$ to give us l_4, u_4 and l_8, u_8 respectively.

	Number of Packets	
Bound	$p=4$	$p=8$
Lower	20	18
Upper	76	138

Table 4-12: LANSA 4 - Communication Costs

Using the definition of c_p, $c_4=-1.77$ and $c_8=-0.40$ which suggests that the algorithm does not exhibit any communication parallelism. This is probably due to the congested exchange phase where all processors communicate with one another at the same time, thus resulting into a lot of packet collisions.

4.5.6. Summary

In this section, we summarize the communication parallelism attained by each algorithm. Table 4-13 shows the communication parallelism and the communication costs (in milliseconds) for each algorithm for $p=4$ and $p=8$.

Algorithm	t_4	t_8	c_4	c_8
LANSA 0	1096	1841	0.43	-0.03
LANSA 1	3035	5289	0.04	0.37
LANSA 2	2281	4271	0.49	0.50
LANSA 3	1440	4080	0.78	0.92
LANSA 4	2802	2984	-1.77	-0.40

Table 4-13: Communication Parallelism and Communication Costs

Except for LANSA 0, each LANSA attained higher communication parallelism when the number of processors was increased. This is a surprising result. We expected the communication parallelism to decrease as the number of processors increased because more processors are contending for access to the Ethernet and the rate of packet interference should increase. This suggests that out theoretical model is incomplete. A more sophisticated model should consider packet collisions and buffer overflows since these two factors appear to be the source of the problems encountered by our algorithms.

Table 4-13 also shows that the communication cost is not linearly related to the communication parallelism.

Chapter 5

Conclusion

This research provides us with empirical results for selected parallel and distributed sorting algorithms in the local area network environment. We name these algorithms Local Area Network Sorting Algorithms. We adopted a cost model which accounted for both communication costs and local processing costs. The experimental results reported in this thesis confirm that our cost model is correct; both cost parameters -- communication costs and local processing costs, are important in evaluating the LANSAs.

The bulk of the results of this thesis concern the performance of the algorithms that we have implemented. The two main parameters that were varied during the experiments were N (number of keys to sort) and p (number of processors available for sorting). Various sizes of data sets were sorted successfully on four and eight processors.

Normally, the performance of a sorting algorithm is dependent on the size of the data set, i.e., the larger the data set, the longer the algorithm takes to sort the keys. The individual LANSAs behave normally except in the case of LANSA

3 for $p=8$. A possible reason for the unexpected behaviour of this algorithm is that the value of T (termination criterion) we used was not suitable for all values of N. More tests with different values for T would tell us whether the choice of T is causing the unexpected behaviour of this algorithm.

We ranked the algorithms for four and eight processors assuming N is large. LANSA 0 is the worst algorithm of the five LANSAs for $p=4$ and $p=8$. This result was expected since this algorithm does not take advantage of the distributed nature of the local area network environment. LANSA 2 is the most efficient algorithm for $p=4$ while LANSA 4 is the most efficient algorithm for $p=8$.

We analyzed the local processing costs and communication costs of the LANSAs for four and eight processors assuming N is large. The local processing costs are lower for $p=8$ than for $p=4$ because the size of the local file at each processor is smaller in the former case. The communication costs are higher for $p=8$ than for $p=4$. We think that this is because there are more processors contending for access to the Ethernet, so the number of packet collisions is greater.

We extrapolated the behaviour of the LANSAs assuming that the performance of the network is improved. Improvement of network performance is possible if we use a performance oriented interprocess communication facility such as the *V Kernel* instead of the Sun UNIX interprocess communication facility. However, the *V Kernel* was not available at the time of the experiments. Nevertheless, our extrapolations suggest that if the network performance is improved by 90%, LANSA 1 (LANSA 2) becomes the overall best algorithm when run on four (eight) processors.

We analyzed the performance of the algorithms for $p=4$ and $p=8$ with constant N and compared the results. One reason for doing this kind of analysis was to find out whether the theoretical results agree with the experimental results. The other reason was to determine whether any kind of communication parallelism can be achieved by the different algorithms and to study the way that communication parallelism changes as the number of processors increases. We found that the theoretical predictions of the local processing costs were approximately equal to the experimental results. To determine if an algorithm achieves communication parallelism, we developed a framework within which the degree of communication parallelism can be estimated. We found that some parallelism is achieved during the packet building and buffer management processes. The parallelism is not entirely dependent on the Sun UNIX networking software. It also depends on the nature of the transmission phases of each algorithm. LANSA 1 and LANSA 2 exhibit some communication parallelism while LANSA 4 exhibits no parallelism. LANSA 0 exhibits no parallelism for eight processors; however, parallelism was achieved for four processors. LANSA 3 appears to achieve a high degree of communication parallelism.

The amount of communication parallelism achieved by the different algorithms was not as anticipated. This is probably because our theoretical model does not consider packet collisions and buffer overflows. A more sophisticated model which accounts for packet collisions and buffer overflows should give more accurate predictions of communication parallelism.

Appendix A

Fundamental Parallel Merging Schemes

In this Appendix, two algorithms are discussed: Odd-Even Transposition Sort and Stone's Bitonic Sort. These two algorithms have been generalized to block sorting algorithms which have been adapted to our model.

A.1. Odd-Even Transposition Sort

The serial Odd-Even Transposition Sort is a variation of the *bubble sort*. It requires n phases, each of which requires $n/2$ comparisons (n is the number of elements to be sorted). Odd and even phases alternate; during an odd (even) phase, odd (even) elements are compared with their right adjacent logical neighbor, i.e. the pairs (x_1,x_2), (x_3,x_4) . . . $((x_2,x_3)$, (x_4,x_5) . . .) are compared. To obtain a completely sorted sequence, a total of n phases is required [BDHM 84].

This serial algorithm can easily be parallelized. Consider n processors $P_1,P_2,\ldots,P_n$. Assume that initially x_i resides in P_i for $i=1,2,\ldots,n$. To sort $(x_1,x_2,\ldots,x_n)$ in parallel, let $P_1,P_3,P_5,\ldots$ $(P_2,P_4,\ldots)$ be active during the odd (even) steps, and execute the odd (even) phases of the serial odd-even transposition sort in parallel. Thus, the parallel odd-even transposition algorithm sorts n numbers with n processors in n parallel comparisons and $2n$ transfers. An example is illustrated in Figure A-1 for $n=4$.

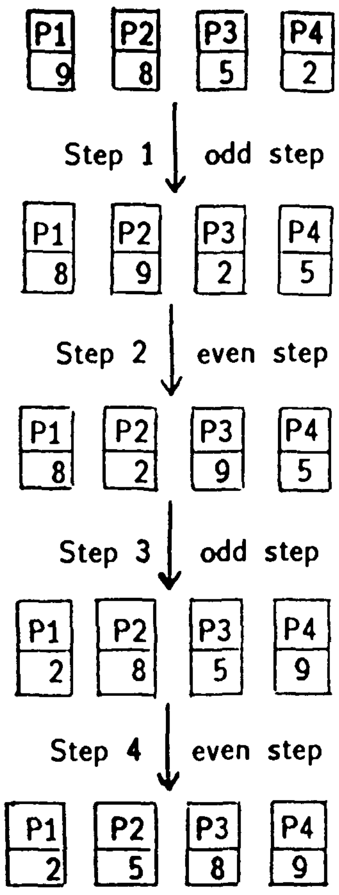

Figure A-1: Parallel Odd-Even Transposition Sort

A.2. Stone's Bitonic Sort

Stone's Bitonic Sort is an algorithm that utilizes an interconnection pattern called the perfect shuffle. A **perfect shuffle** of elements of a vector is similar to shuffling a deck of cards so that after a shuffle the elements from the two halves of the vector alternate. To make a sorting network fast, it is necessary to have a number of comparators perform comparisons in parallel. A **comparator** is a comparison-exchange module that receives two numbers on its two input lines A,B and outputs the minimum on its line L and the maximum on its output line H [Sto 71].

To sort n numbers, Stone's bitonic sort requires a total of $(n/2)(\log n)^2$ comparators, arranged in $(\log n)^2$ ranks of $(n/2)$ comparators each. The network has $(\log n)$ stages, with each stage consisting of $(\log n)$ steps. At each step, the output lines are shuffled before they enter the next rank of comparators. The comparators in the first $(\log n - i)$ steps do not exchange their inputs; their only use is to shuffle their input. The network that realizes this algorithm is shown in Figure A-2 for eight input lines [BDHM 84]. For sixteen input lines, the network connection can be found in [Knu 73].

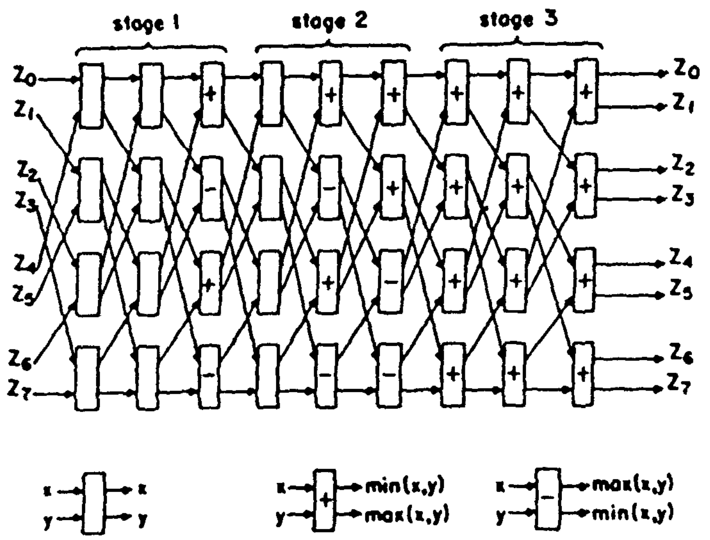

Figure A-2: Stone's Bitonic Sort

Appendix B

Empirical Results

In this Appendix, we tabulate the empirical results obtained during our experiments. The results are shown in tables B-1 and B-2 for $p=4$ and $p=8$ respectively. These results are also plotted in figures 4-1 and 4-2 respectively. Tables B-3 and B-4 show the breakdown of the costs components for $p=4$ and $p=8$ respectively.

			N	
Algorithm	$2K$	$4K$	$8K$	$16K$
LANSA 0	1.358	2.947	6.827	14.546
LANSA 1	0.766	1.780	3.671	7.287
LANSA 2	0.920	1.637	3.565	6.667
LANSA 3	1.360	2.280	4.020	8.240
LANSA 4	1.036	2.058	3.620	7.852

Table B-1: Response Time (in seconds) for $p=4$

			N	
Algorithm	$2K$	$4K$	$8K$	$16K$
LANSA 0	1.367	4.431	7.535	14.984
LANSA 1	0.986	1.971	4.194	7.873
LANSA 2	1.182	1.591	3.161	6.758
LANSA 3	5.560	5.100	6.540	7.380
LANSA 4	1.337	2.027	3.430	6.294

Table B-2: Response Time (in seconds) for $p=8$

Cost	N 2K	4K	8K	16K
Local Proc.	1.246	2.723	6.266	13.450
Comm.	0.112	0.224	0.561	1.096

(a) LANSA 0

Cost	N 2K	4K	8K	16K
Local Proc.	0.452	1.035	2.014	4.252
Comm.	0.314	0.745	1.657	3.035

(b) LANSA 1

Cost	N 2K	4K	8K	16K
Local Proc.	0.472	0.996	2.150	4.386
Comm.	0.448	0.641	1.415	2.281

(c) LANSA 2

Cost	N 2K	4K	8K	16K
Local Proc.	0.640	1.440	3.040	6.800
Comm.	0.720	0.840	0.980	1.440

(d) LANSA 3

Cost	N 2K	4K	8K	16K
Local Proc.	0.661	1.236	2.987	5.050
Comm.	0.375	0.822	0.633	2.802

(e) LANSA 4

Table B-3: Cost Components (in seconds) for the LANSAs for $p=4$

Cost	2K	4K	8K	16K
			N	
Local Proc.	1.248	2.744	5.920	13.143
Comm.	0.119	1.687	1.615	1.841

(a) LANSA 0

Cost	2K	4K	8K	16K
			N	
Local Proc.	0.275	0.592	1.364	2.584
Comm.	0.711	1.379	2.830	5.289

(b) LANSA 1

Cost	2K	4K	8K	16K
			N	
Local Proc.	0.253	0.541	1.174	2.487
Comm.	0.929	1.050	1.987	4.271

(c) LANSA 2

Cost	2K	4K	8K	16K
			N	
Local Proc.	0.360	0.880	1.860	3.300
Comm.	5.200	4.220	4.680	4.080

(d) LANSA 3

Cost	2K	4K	8K	16K
			N	
Local Proc.	0.507	0.863	2.073	3.310
Comm.	0.830	1.164	1.357	2.984

(e) LANSA 4

Table B-4: Cost Components (in seconds) for the LANSAs for $p=8$

References

[Baa 78] Baase, S.
Computer Algorithms: Introduction to Design and Analysis.
Addison-Wesley Publishing Co., Reading, Mass., 1978.

[BDHM 84] Bitton, D., Dewitt, D.J., Hsaio, D.K. and Menon, J.
A Taxonomy of Parallel Sorting.
ACM Computing Surveys 16(3), September, 1984.

[BrH 83] Broomell, G. and Heath, J.R.
Classification Categories & Historical Development of Circuit Switching Topologies.
ACM Computing Surveys 15(2), June, 1983.

[Che 84] Cheriton, D.R.
The V Kernel: A Software Base for Distributed Systems.
IEEE Software. 1(2), April, 1984.

[Knu 73] Knuth, D.E.
The Art of Computer Programming: Sorting and Searching.
Addisson-Wesley Publishing Co., Reading, Mass., 1973.

[MeB 76] Metcalfe, R.M. and Boggs, D.R.
Ethernet: Distributed Packet Switching for Local Computer Networks.
ACM Communications 19(7), July, 1976.

[PaP 85] Page Jr., T. W. and Popek, G.J.
Distributed Data Management in Local Area Networks.
In *Proc. 3rd ACM Symp. on Princ. of Database Systems.*, ACM-SIGACT-SIGMOD, March, 1985.

[RSS 85] Rotem, D., Santoro, N. and Sidney, J.
Distributed Sorting.
IEEE Trans. on Comp. C-34(4), April, 1985.

[SaS 82] Santoro, N. and Sidney, J.B.
Communication Bounds for Selection in Distributed Sets.
Technical Report SCS-TR-10, School of Computing Science, Carleton University, September, 1982.

[SaS 83] Santoro, N. and Sidney, J.B.
A Reduction Technique for Selection in Distributed Files: I.
Technical Report SCS-TR-23, School of Computing Science, Carleton University, April, 1983.

[SFR 83] Shrira, L., Francez, N. and Rodeh, M.
Distributed k-Selection: From a Sequential to a Distributed Algorithm.
In *Proc. 2nd ACM Symp. Princ. Distrib. Comput.*. ACM-SIGACT, August, 1983.

[Sta 84] Stallings, W.
Local Networks.
ACM Computing Surveys 16(1), March, 1984.

[Sto 71] Stone, H.S.
Parallel Processing with the Perfect Shuffle.
IEEE Trans. on Comp. C-20(2), February, 1971.

[Svo 84] Svobodova, L.
File Servers for Network-Based Distributed Systems.
ACM Computing Surveys 16(4), December, 1984.

[Weg 84] Wegner, L.M.
Sorting a Distributed File.
Computer Networks (8), August, 1984.