

INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.
2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.
3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.
4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.
5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

**University
Microfilms
International**

300 N. Zeeb Road
Ann Arbor, MI 48106

8504325

Kapadia, Rajiv J.

A THEORETICAL COMPARISON OF FOUR PARALLEL PROCESSING
NETWORKS

The University of Oklahoma

PH.D. 1984

University
Microfilms
International 300 N. Zeeb Road, Ann Arbor, MI 48106

Copyright 1985

by

Kapadia, Rajiv J.

All Rights Reserved

THE UNIVERSITY OF OKLAHOMA
GRADUATE COLLEGE

A THEORETICAL COMPARISON OF FOUR
PARALLEL PROCESSING NETWORKS

A DISSERTATION
SUBMITTED TO THE GRADUATE FACULTY
in partial fulfillment of the requirements for the
degree of
(DOCTOR OF PHILOSOPHY)


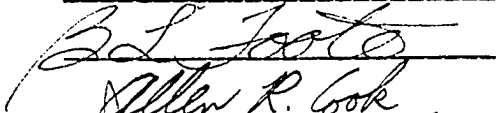
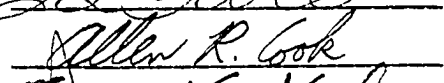
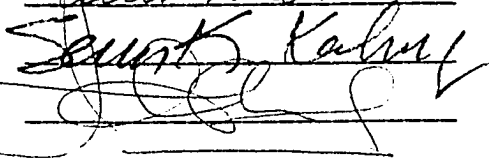
BY
RAJIV J. KAPADIA
Norman, Oklahoma

1984

A THEORETICAL COMPARISON OF FOUR
PARALLEL PROCESSING NETWORKS
A DISSERTATION

APPROVED FOR THE SCHOOL OF ELECTRICAL ENGINEERING

BY

© 1985

RAJIV J. KAPADIA

All Rights Reserved

TABLE OF CONTENTS

	Page
LIST OF TABLES	iv
LIST OF ILLUSTRATIONS	v
LIST OF GRAPHS	vi
Chapter	
I. WHY ANALYZE ALGORITHMS	1
II. A GENERAL COMMUNICATION ORIENTED MODEL	5
III. COMMUNICATION OF DATA IN A MULTIPROCESSING SYSTEM	13
IV. PERFORMANCE MEASURES	29
V. USING THE PERFORMANCE MEASURES	46
VI. CONCLUSIONS	73
OTHER RESEARCH DIRECTIONS	78
APPENDIX	80
BIBLIOGRAPHY	86

LIST OF TABLES

TABLE	Page
1. Summary of parallel computation results	13
2. Performance measures	45
3a. Communication time to add a vector on various networks	50
3b. Comparisons of various networks to add a vector for different numbers of processors in a network	52
4. Algorithm setup to multiply two matrices	53
5. Communication time to multiply two matrices..	58
6. Comparisons of various networks to multiply two matrices for different numbers of processors in a network	60
7. Communication time to sort an array of numbers	66
8. Comparisons of various networks to sort an array with different numbers of processors in the network	68
9. Mean distances between processors	71

LIST OF ILLUSTRATIONS

	Page
1. THE RING NETWORK THE TREE NETWORK	12
2. A THREE GENERATION TREE NETWORK	20
3. SORTING ON A MESH CONNECTED NETWORK WITH DIFFERENT INDEXING	26

LIST OF GRAPHS

GRAPHS	Page
1. The communication time to add a vector on various networks	51
2. The communication time to multiply two matrices	59
3. The communication time to sort an array on different networks	67

ABSTRACT

Previous work on the analysis of execution time of parallel algorithms has either largely ignored communication delays or has dealt with specific interconnection structures such as the perfect shuffle and the nearest neighbor. In this paper it is shown that the communication time is just as significant as the execution time and that the communication time is dependent upon the data size. Four networks are compared, using parameters that are defined in the paper. Using a few representative algorithms it is determined that the communication time depends on, (1) the average distance between processors when the number of processors in the network is large and, (2) the average number of processors a given processor is connected to when the number of processors in the network is small. The breakeven point varies from algorithm to algorithm.

A THEORETICAL COMPARISON OF FOUR PARALLEL PROCESSING NETWORKS

CHAPTER I

WHY ANALYZE ALGORITHMS?

In 1966, Flynn, in his classical paper, "Very High Speed Computing Systems (3)," introduced us to the concept of multiprocessing. One of his proposals, the single instruction multiple data machine, is gaining a lot of importance. This is due to the advent of VLSI technology and the fact that such a machine has been built and found useful. The Illiac IV is an example of such a machine.

As understanding of the machine grows, the properties and problems in the machine hardware and software are being identified. Until recently, most design engineers were of the belief that problems in the machine were either hardware related or software related. It is now realized that proper matching of software with hardware is essential. One of the factors governing the execution time of an algorithm is the time used moving data on hardware as required by the algorithm. This underscores that hardware and software are not two distinct domains. They are a blend. It is necessary to have a proper match between the two.

Communication in a multiprocessing environment can become a major problem. Properly designed parallel

structures that need to communicate with their nearest neighbor will gain the most from a well thought out match between the algorithm and the interconnection structure. Precious time and performance are lost when modules that are far apart must communicate. For example, if data are to be sent to a processor other than the nearest neighbor, two or more data transfer cycles will be necessary. The design engineer must have this communication bottleneck uppermost in mind when evaluating possible structures and interconnect networks to implement an algorithm.

The study of communication time for information flow in a network is not a new problem. For decades letter carriers have tried to find the shortest path for their appointed route. The study of communication time for data in a multiprocessing system is relatively new. L. Ford and D. Fulkerson are among the leaders in this study. Their initial contribution was in 1962 when they studied flows in networks.

S. Bokhari [8] published a paper in 1979 which showed one way to schedule two processors with dynamic re-assignment.

B. Lint in 1979 has shown that in a multiprocessing system the time used by a data element moving from one processor to another is just as important as the time used by a data element to compute a result.

In 1978 M. Gentelman [5] presented a paper which dealt with minimizing the communication time for some matrix operations on the Illiac IV.

In 1981 B. Lint and T. Agerwala [17] talked about design and analysis of parallel algorithms. Their talk emphasized the importance of communication time in a multiprocessing system.

Efforts of these people show that the study of communication time in a multiprocessing system is of growing concern to both the designers and users alike. Most of the recent advances made in the study of communication time treat the algorithm as a finite state automata or as a sequential machine. While these analysis procedures are not without merit I think they are treated out of context. A sequential machine has one active state and the next state depends on the input signal and the current state. This is not so in a multiprocessing system where, due to local control, some of the processors may ignore or modify a broadcast instruction.

It is my belief that since the algorithm has to execute on the network, the communications required by the algorithm must be those that are most suited to the network. In other words, if the network and the moves required by the algorithm match, then the communication time will be less than any other combination of the network and the algorithm.

Thus, with communication of data in a multiprocessing system being one of the key factors affecting the execution time of an algorithm, I propose to present a method of matching an algorithm to a network. In the first section I will present a general communication oriented model. This model, out of necessity, will be a very general model. Any particular points will be discussed as, and when necessary in the course of the presentation. In the next section, using the model, a lower bound on the execution time is obtained. Following this, some factors will be found that affect the communication time in such a way that the lower bound obtained earlier is not always achieved. These same factors, and the way they affect the communication time, prevent the writing of a mathematical formula for communication time. Next I will show some approximations that can be used, allowing us to obtain performance measures. These can be used to measure how well an algorithm and a network match. Following this will be some examples that show how these performance measures can be used to study the match between algorithm and the interconnect structures.

CHAPTER 2

A GENERAL COMMUNICATION ORIENTED MODEL

In this section several models for parallel processing are discussed. A model of parallel processing should include all the factors that affect the execution of an algorithm on a network. This necessitates the inclusion of computation as well as communication. The model will provide a foundation for the design methods. Using this model, practical multiprocessing systems can be designed. The models presented will include communication and computation aspects in varying degrees.

A model for multiprocessing system is basically an extension of a sequential random access machine. This description of a model was made by A. Aho, and others in their book, "The Design and Analysis of Computer Algorithms." In a synchronous model all processors are driven by a single clock. In one clock period all processors that are active execute a single operation. These processors have some local control and hence, the operation can be different for different processors. Some of the operations could be add, store, compare and send.

Even though the operations can be different for different processors, each processor begins and terminates the operation simultaneously. Processors with shorter operations wait.

Now that we have a well defined beginning and termination of each operation, it is possible to define a measure of complexity analogous to the computation complexity measure of a sequential machine. This complexity is the number of steps needed to execute the program as a function of the size of the input.

The Unbounded Model. One common synchronous model assumes K parallel processors. Each processor is identical to each other and has access to the same storage location. Communication of data is done by write-read cycle. A major bottleneck occurs at memory access. Since it is assumed that the computation result of one cycle is available to any processor on the following cycle, the case when the number of processors is arbitrarily large is known as unbounded parallelism. Processor complexity in an unbounded model is equal to the number of processors necessary to execute the algorithm.

The Network Model. The unbounded model ignores communication time by making the processors access the memory. This memory access is assumed to take place in constant time. No consideration is given to communication time. Memory conflicts are considered part of execution

time. A fundamental extension of this model eliminates the common mass memory and computation results are made available to the proper processors via a network of interconnections. Each node on this interconnection scheme is associated with a processor. Communication of information is done by store and send approach.

If the network model is used, the architecture of the multiple processing system would then have all the processors having their own local memory. In this local memory, the processor will be able to store some control routines which the processor can execute. Exchange of information between processors takes place through the network. Logically neighboring processors can exchange information in one cycle via hardware links connecting the processors. Non-adjacent processors exchange data by sending the information from processor to processor using a store and send approach. Information cannot be broadcast from one processor to every other processor in the system, but any one processor can send its data value to all of its logically neighboring processors.

Thus, the crucial part of the design of any highly parallel computing system is the interconnection network. The network consists of relatively standard microprocessors with hardware links interconnecting them. An essential criterion for the network selection is physical feasibility. The network must conform to the practical

building restrictions imposed by electronic technology. A simple way of modelling these restrictions abstractly is to insist that each node of any network be connected to no more than a fixed number C of other nodes, independent of the total number N of nodes in the network. N can become very large.

Algorithm execution on this network proceeds in a sequence of non overlapping cycles. Each cycle consists of an execution phase and a message transfer phase. One phase can start only after the other phase has finished execution on each processor in the system. Either one of these phases could be nil for one or more processors during any cycle. Each one of these transfer phases is further subdivided into periods. During any message transfer phase an active processor can send data to its logically adjacent processor in one period. This can be further routed during the next period if necessary. During any execution phase, the execution of one instruction, like add or compare, is executed. If more operations need to be performed, they are executed on subsequent periods.

The message transfer phase and the execution phase can be of unequal lengths in a cycle. Also the message transfer phases or the execution phases of different cycles can be of unequal lengths. The number of periods will thus, vary from phase to phase and cycle to cycle, but the number of periods executed by all the processors during any

given phase of any given cycle will be the same. As a result of this, all the processors will begin a phase simultaneously and terminate the same phase simultaneously. The message transfer proceeds through the interconnection network with the store and send approach. In this approach a processor receives data in one period. During the following period the processor will decide if the data belongs to it or is to be sent down the line. If it is to be sent down the line, then during the third clock period this data will move to the next processor. If during one period a processor receives two or more data items then one or more will have to wait to be processed and sent forward.

As mentioned earlier all processors have some private memory. This private memory cannot be accessed by other processors in the network. The private memory is used to store variables or constants that the algorithm might need while executing. It is also used to store tables that the processor refers to in order to route data through the shortest conflict free path possible. The processors also have some limited capacity for local control. These control routines are also stored in private memory. In this way, mask generation is done locally. The mask generation routines are necessary since all the processors do not execute every instruction of the algorithm.

As mentioned earlier, execution of an algorithm on such a model proceeds in a sequence of non-overlapping

cycles of computation and communication phases. Of these, which phase is more important? Many attempts have been made at answering this question. A proper answer would be that both the phases are equally important. If careful attention is not paid to communication, then moving data over long paths can eat up all the gains that the algorithm makes by a brilliant execution technique.

In a multiprocessing environment, more often than not, data for the next computation stage is made available during the current computation stage. This data may be at some other processor and must be moved to the proper processing element. In this way, in algorithm execution the computation and the communication phases alternate. The communication takes place over the interconnection network. Careful attention must be paid to the interconnection network and how it moves data around.

As an example, let us look at how a couple of networks perform when asked to add a vector of N elements on $N/2$ processors. The only difference between the two networks is the interconnection network. One system has a tree-like network, while the other one is a ring-like structure. Executing the algorithm on the ring network causes the sum

Function $O(\cdot)$ is used as a timing function. The variable used is the controlling parameter and the time is of the order of magnitude of that variable ex $o(N)$ could be $2N + 5$ or any linear function of N .

to be obtained in $O(N)$ time. This is the same as in a sequential uniprocessor system. On the tree structure, the time to execute the algorithm is $O(\log N)$. This is a definite improvement over the ring structure. Further study of the ring structure (Figure 1), shows that to perform the final sum, the data had to be moved a distance of $N/4$ processors. On the tree structure (Figure 2), the data had to be moved only one unit distance to perform any sum.

The tree structure is known as an asymptotically optimal network for this particular algorithm. An asymptotically optimal network is that network which performs various fundamental computation operations on a collection of N data items as rapidly as theoretically possible. Table 1 shows some operation and networks that should be matched together.

The list in Table 1 and others like that which can be found in other articles are some of the advances made in this field. The various authors in the lists have worked on each individual algorithm on a network they specified. All this effort is dedicated in a very narrow field. It does not help if you do not have a match as the authors have demanded. The work that I am presenting here is more of a general nature. It points the direction a network designer will take when he is trying to build a network.

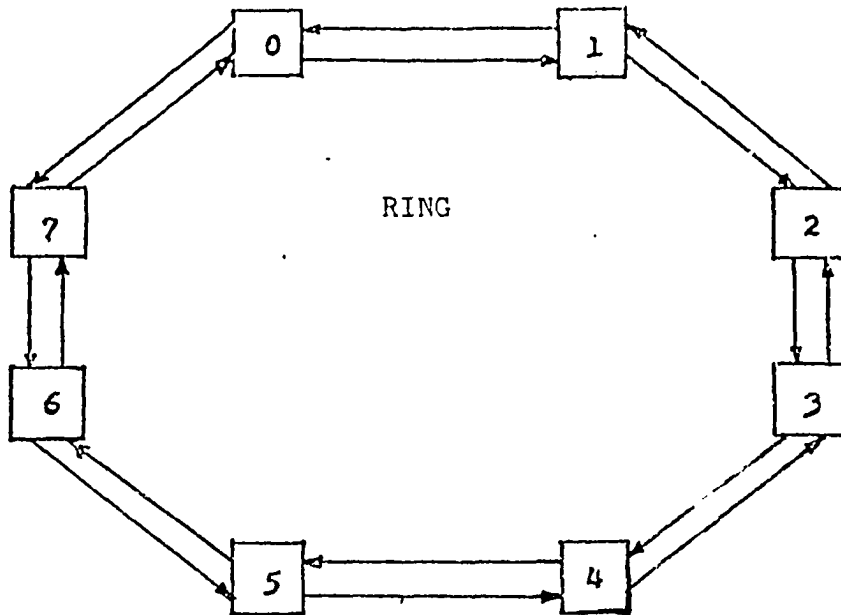


FIGURE 1

- (1) Let $C = 1$
- (2) Add the data elements present in each processor.
- (3) Move data from processor $(2i + C)$ to processor $2i \forall i$
- (4) If $C \geq N/2$, STOP.
- (5) $C = 2 * C$
- (6) Go to step 2.

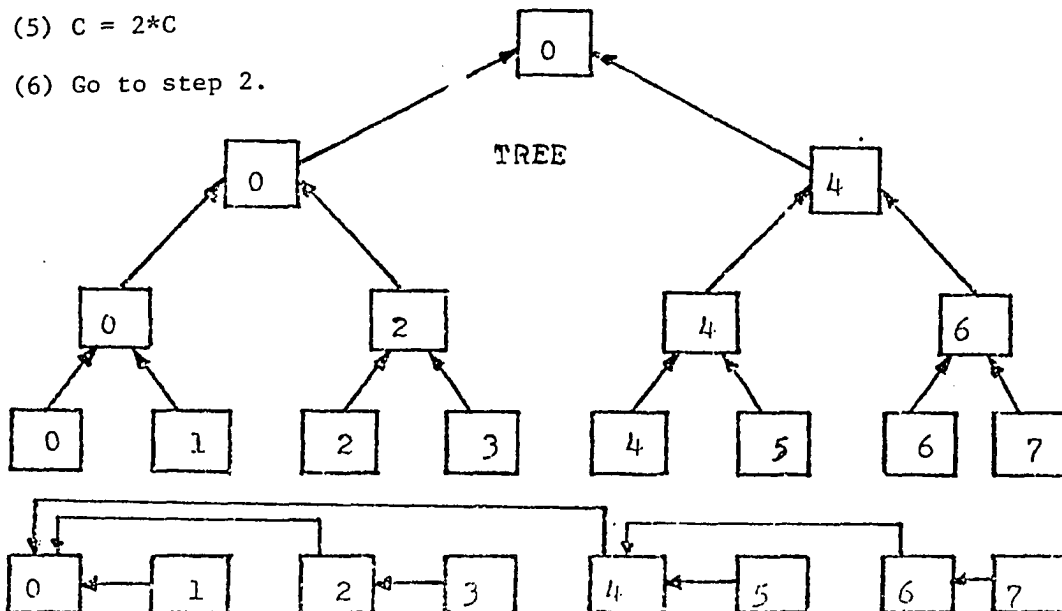


TABLE 1
SUMMARY OF PARALLEL COMPUTATION RESULTS

No.	Process	Parallel Time	Sequential Time	Reference
(1)	All partial sums of N elements	$O(\log N)$	$O(N)$	Pease (1968) Stone (1971)
(2)	Merging two lists of N items	$O(\log N)$	$O(N)$	Batcher (1968)
(3)	Sorting N items	$O(\log^2 N)$	$O(N \log N)$	Batcher (1968)
(4)	Fast Fourier trans- forms	$O(\log N)$	$O(N \log N)$	Pease (1968) Stone (1971)
(5)	Inversion of mat- rices of size $N^{0.356}$	$O(\log^2 N)$	$O(N^{3/2.81})$	Csanky (1975)
(6)	Multiplications of matrices of size ($N \log N \log \log N$)	$O(\log N \log \log N)$	$O(N \log N \log \log N)$	Chandra (1976)

In 1978 at the International Computer Conference, T. Agerwala showed that it is possible to devise different amounts of communication time and computation time by modifying the algorithm itself. With this knowledge the designer may be able to trade communication time with computation time and arrive at an execution time which is the smallest from the options he has considered. Agerwala suggested that if this is done systematically, then the effective execution time can be reduced. As an example of this Agerwala cited a sequential algorithm that requires $O(N^2)$ computation steps and no communication steps to obtain a FFT, then after some trades are made a similar algorithm is run on an N processor system, the computation steps are reduced to $O(N \log_2 N)$ while the communication time is increased to $O(N \log_2 N)$. This is a saving in the overall execution time for the algorithm.

The distance through which data items have to travel to reach their destination is another factor that affects communication time. Properly designed parallel structures that need to communicate with only their nearest neighbors will gain the most from a well thought out design. One reason is very obvious, the shorter the path, the quicker it is travelled. Another affect of long data paths is rather subtle. If data items travel over long distances, then there is a greater probability of having data path conflicts. Moving data without data path conflicts is one

goal designers try to achieve. Whenever there is a data path conflict, one of the data items will be delayed. This delay increases the communication time.

CHAPTER 3

COMMUNICATION OF DATA IN A MULTIPROCESSING SYSTEM

In the previous section we saw that the communication time in the execution of an algorithm depends on how data is moved from one processor to another. This data can be moved in three different ways. These ways are 'Store and Retrieve', 'Shared Bus' and 'Message Transfer through an interconnection network'.

Store and Retrieve. In this method the data item is placed in a memory that can be accessed by all the processing elements and whichever processor needs the data item can access the location and obtain the value. A major drawback in this method is that all the active processors will compete, first to get memory access to place their data items into the memory and then, once again, to gain access to obtain the data item they have to use during the next computation cycle.

Shared Bus. In this method all the processors are connected to a bus system. These can be single or multiple buses. A processor places its data on the bus which is available to all the processors. The destination processor

can grab the data while the rest ignore it. The major drawback to this method is the competition for the bus. During the data transfer stage all the processors will try to get control of the bus.

Message transfer through an interconnection network.

In this method, each processor is connected to a fixed number of other processors and can transfer data only to these processors. One of these processors will grab the data and transfer it further along the path to the destination processor. The major drawback in this method is the data path conflicts that can occur at the processors. This conflict occurs when a processor receives data from two of its neighboring processors at the time.

A comparison of the three methods will show why I chose the interconnection network as the message transfer method to use in the previous and the following sections. To make this comparison, let us assume that there are a large number of processors in the network. Also let us assume that one half of them are going to send data to other processors. By a large number is meant at least sixty-four processors.

Lemma 1: Under the conditions described above, 'Shared Bus' message transfer is more efficient than 'Store and Retrieve'.

Proof: Say that cN of the N processors have to transfer data to other processors. ($c \approx 1/2$). Of these cN

processors, let m processors transfer data by shared bus, while $(cN-m)$ processors transfer by 'Store and Retrieve'.

Case 1. $cN-m \geq m$

In the first m periods, let m data items be transferred by shared bus. At the same time, let m of the $cN-m$ data items be placed in memory for 'Store and Retrieve'. After these m periods, there will be $cN-2m$ data items left over, to be placed in memory. This can be done in $cN-2m$ periods. Next, the data items placed in memory can be retrieved in $cN-m$ periods since that many data items have been placed in memory. Thus the total time for the message transfer stage is: $T(cN, m) = m + [cN-2m] + cN-m = 2cN-2m = 2(cN-m)$.

Case 2. $cN-m \leq m$

This time, in the first $cN-m$ periods let $cN-m$ items be placed in memory and $cN-m$ data items be broadcast. Thus in the next $m-(cN-m)$ cycles, the remaining data items can be broadcast. At the same time $m-(cN-m)$ data items can be read back from the memory. Still, there are $cN-m-[m-(cN-m)]$ data items in memory which have to be retrieved. These can be obtained in $cN-m-[m-(cN-m)]$ more cycles. The total time necessary is $T(cN, m) = cN-m + m-(cN-m) + cN-m-[m-(cN-m)]$. $T(cN, m) = 2cN-2m = 2(cN-m)$.

Either way the message transfer is done the time required is the same. Now:

$$T(cN, m+1) - T(cN, m) = 2cN-2(m+1) - (2cN-2m)$$

$$T(cN, m+1) - T(cN, m) = -2$$

The negative result shows that increasing the number of transfers by shared bus, will decrease the number of periods in a message transfer stage. Therefore 'Shared Bus' is more efficient than 'Store and Retrieve'. Similarly, with the same conditions, it can be shown that no advantage is gained by 'Shared Bus' message transfer as compared to 'Message Transfer through an interconnection network'.

In performing a transfer through a network, the analysis is not so simple. One of the factors that plays an important part is the distance a data item has to travel through. It is necessary to define a distance function for each network.

Definition 1: The number of processors (or nodes) through which a data item has to go to reach its destination is the distance the data item has to travel. Some of the properties of the distance function are:

- (1) $d(i,j) \leq d(i,k) + d(k,j)$
- (2) $d(i,j) > 0$
- (3) $d(i,i) = 0$
- (4) $d(i,f(i)) = 1$

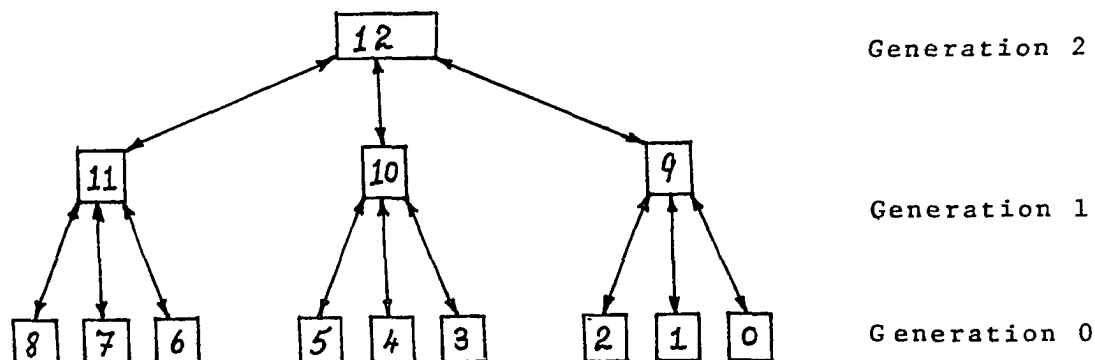
Here $f(i)$ represents the logically adjacent processor to processor i in the interconnection network that is being studied. With these properties, we can compute the distance function in general for some interconnection structures.

The Ring network. $d_R(i,j) = \min(|i-j|, N-|i-j|)$ and $\max d_R(i,j) = N/2$.

The Grid network. Let $m = \min(|i-j|, N-|i-j|)$ then, $d_G(i,j) = \lceil \frac{m}{\sqrt{N}} + 0.49 \rceil + |m - \sqrt{N} \lceil \frac{m}{\sqrt{N}} + 0.49 \rceil|$ and $\max d_G(i,j) = \sqrt{N}$.

The Shuffle-Exchange network. Let $m = \log_2(N)$ $d_{SE}(i,j) = m + \text{the number of bits in which } i \text{ differs from } j$ and $\max d_{SE}(i,j) = 2m$.

The Tree network. A tree network is built by connecting the K roots of K level $i-1$ trees to one common root. This gives a level i tree network. A level 0 tree network is one root by itself. In any tree which is level 1 or more, each node has 1 father, $K-1$ brothers and K sons. A node can communicate directly with its father or all of its K sons. The following figure shows a 2 level tree with three sons per father.



The number of processors in a tree is given by

$$N = \sum_{x=0}^i K^x = \frac{K^{i+1} - 1}{K - 1} \quad K = \# \text{ sons}$$

$i = \# \text{ of generations}$

In this network a message can be transferred in one step going from son to father $[(\frac{K-1}{K})N] + [x/K]$

Then going from father to son $K*x - (K-1)*N + \{-1, 0, 1, 2, \dots, K-2\}$. Thus in one step the tree network can transfer data a max distance of $(\frac{K-1}{K})N$ processors.

So: $D_T(i, j) = 2 * \text{number of levels in the tree} - \text{level of processor } i + \text{level of processor } j$.

And: $\max d_T(i, j) = 2 * \text{number of levels in the tree} - 1$ or $2 * [\sqrt[K]{N}]$.

From the distance functions we see that the ring connected network will take the longest to transfer data from processor i to j . Thus, if a ring connected network is more efficient than 'Shared Bus', then all the other networks will be more efficient. (It can be shown through exhaustive search that the ring connected network has the largest max distance function of all the other networks used to transfer data).

Lemma 2: Under the conditions described earlier, message transfer through the network is more efficient than shared bus transfers.

Proof: Say that cN of the N processors have to transfer data ($c \approx 1/2$). Of these cN let m processors transfer data by message transfer, while $cN-m$ transfer through a shared bus. It will take at most, $N/2$ periods to transfer data by message transfer irrespective of what m is, as long as ($c \leq 1/2$). In this way the message transfer

can be accomplished in $N/2$ periods. Thus, $N/2$ is a lower bound on the message transfer stage. Now, suppose $c > 1/2$. Even for $c > 1/2$, all the data items can be transferred in $N/2$ periods through message transfer. If broadcasting all the items is tried, then, cN periods will be required. Since $c > 1/2$ more than $N/2$ periods will be necessary. This shows that ring network and broadcasting require the same number of periods when $c = 1/2$ and the ring network wins out when $c > 1/2$. Generalizing this result, we can say that the breakeven point between a network and message broadcast occurs when the number of items to be transferred equals the max distance on the network. Since the max distance in all the networks that can be used in multiprocessing systems is definitely less than $N/2$, the message transfer should occur through the network and not by broadcast. This is because of our assumption that $c \approx 1/2$.

Now that we know that message transfer through the interconnection structure is better than any other means of transferring data in a highly parallel computer consisting of many relatively standard microprocessors, we want to examine the factors that affect the message transfer operation in a multiprocessing systems. Cost is a factor. This cost factor is the cost of communication which is part of the execution cost. Therefore, a choice of appropriate architecture to transfer messages is important, since

different architectures will have different communication costs. Some of the factors that affect the communication cost are:

- (1) The interconnection structure
- (2) The distance from processor i to j on the interconnection structure
- (3) The number of link contentions encountered
- (4) The movement of data required by the algorithm.

The Interconnection structure. In this effort I will examine four interconnection structures.

- (a) The two way ring: This structure is defined as:

$$R(x) = (x+1) \bmod N$$

- (b) The two way grid network is defined as:

$$G_1(x) = (x+1) \bmod N \text{ and } G_n(x) = (x+n) \bmod N \text{ where } n = \sqrt{N}$$

- (c) The two way shuffle exchange network is defined as; $S(x) = 2(x) \bmod N$

$$E(P_{m-1}, P_{m-2}, \dots, P_1, P_0) = P_{m-1}, P_{m-2}, \dots, P_1, P_0$$

- (d) The two way tree network as it has been described.

These are the networks that will be examined. These are the common networks and any other network of interest can be analyzed the same way as these.

2. The distance from processor i to j on the interconnection structure. In the previous section, we

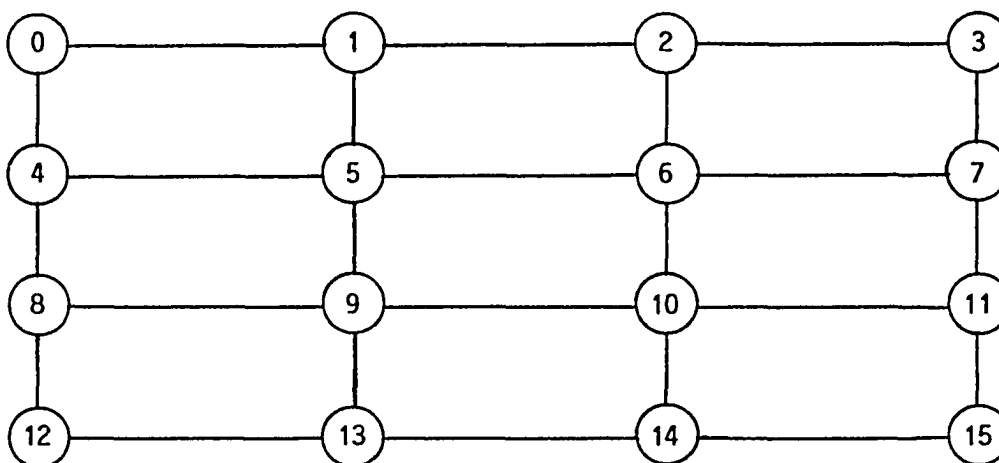
defined the distance function for each of these four networks obtained and the max distance in the network. This distance represents the time it would take to move data from processor i to j on the network. This max distance, which is the time for the message transfer cycle, gives the lower bound of a random move on the particular network. This is not the exact time because (a) All the moves in almost all algorithms have some symmetry and are not random. The effect of this is to reduce the number of periods necessary in a message transfer stage. (b) Often two or more data items will have to compete for a data path or node. The effect of these link contentions is to increase the number of periods in a message transfer stage.

3. The number of link contentions. The interconnection network that we are examining has two-way links and in each of the networks all the processing elements are connected to at least two other processors. It is very possible that some processors may receive a data item from their two or more neighboring processors in one period. Whenever this occurs you have a link contention and one of the data items will be delayed. The number of link contentions is difficult to evaluate since they depend on: (a) The move in the algorithm. Algorithms for different applications require different movement of data to execute. Because these moves are not known before executing the algorithm, it is not possible to obtain these

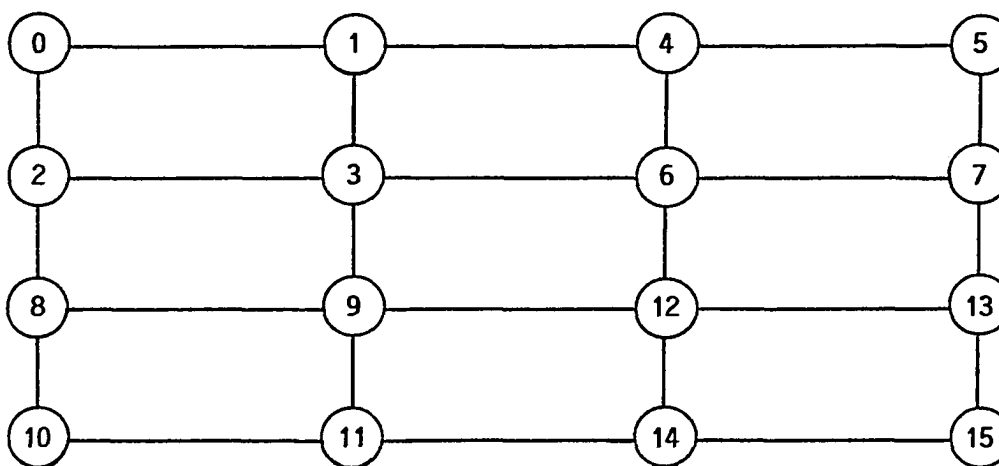
in mathematical form. (b) How these conflicts are resolved. At design time, the design engineer has a few options that can be used to resolve conflicts. Depending on the approach taken by the engineer, the periods necessary for the message transfer stage will vary. The most common approach is to let one of the conflicting data items wait its turn. In this way, the number of periods will be increased by the maximum number of data path conflicts any one data item will encounter.

4. The movement of data required by the algorithm.

All algorithms for multiprocessing systems require that some movement of data items take place. Doing this in such a way that most of the moves are to processors that are close by will reduce communication time. In some examples, it has been shown that indexing processors in a different way can save on communication time. For example, the problem of sorting on Illiac-IV (grid connections) is examined by C. Thompson and H. Kung [9]. An $O(N \log_2 N)$ sorting algorithm based on Batcher's bitonic sort was the best known algorithm prior to this work. The algorithm by Thompson and Kung decreases the execution time to $O(N)$. An important aspect of the $O(N)$ algorithm is that the indexing of processors is different than that used previously and requires less data movement during the sort, (see Figure 3). In each algorithm the ~~i-th~~ smallest data item must be



MESH CONNECTED SORT - IDENTITY INDEXING



MESH CONNECTED SORT - IMPROVED INDEXING

FIGURE 3

moved to processor number i , where i is defined by the indexing scheme.

From the above factors we learn that the distance from processors i to j is a fixed quantity which cannot be changed once the multiprocessing system is built. The movement of data required by the algorithm requires an analysis of individual algorithms. The subject of link contentions has not been studied previously by anyone. This is because it is impossible to determine the exact message conflict without tracing the execution of the algorithm. A delay at one link impacts the delay at other links. This in turn impacts the delay at other links. The pattern of these delays cannot be determined without analyzing the communication network as a whole. This is a very complex problem if one is considering each individual message.

One approach taken is to compute the average delay based on the number of links and the traffic in the network or based on simulation. Simulation may be the most accurate approach but does not fit with an analytic structure of the model.

The approach that I propose is based on the fact that algorithms that specialize in solving large, computationally intensive problems tend to be phased. Execution of each phase requires repeated use of the same move on the network a number of times. The analysis of a

certain type of a move on a network can be carried out. The link contentions for this type of a move can be determined.

In fact, during the design process the concern for communication costs brings up an important point. The choice of an appropriate architecture for an interconnection network is very closely related to the moves necessary in the algorithm. Similarly the choice of a move in an algorithm is closely related to the interconnection networks available. Moving data without data path conflicts is one goal designers try to achieve. Even more important than this is being able to move data to its final destination in one move if possible. If both these conditions are satisfied, then a perfect match between algorithms and the interconnection network has been found.

However, as mentioned earlier, algorithms run in phases and the moves required by different phases are not identical. Thus, a perfect match is almost impossible. If that is so, how close to a perfect match can an algorithm come on a network? Some measures that show how good a match is possible are necessary. The next chapter explores this idea.

CHAPTER 4

PERFORMANCE MEASURES

As we saw in the previous section, the designer's goal is to construct an algorithm that would execute moves on the network which are conflict free and which require only one period to complete. If these goals are achieved, then this is the best match between the algorithm and the network and no improvement is possible. In most algorithms these goals are not achieved because of a couple of reasons. First, the algorithm runs in phases. Each phase requires a different move. Since the network is fixed, it may not be able to execute each move according to the design goals. Second, the algorithm must be executed on a network that is available. This may not be the network for which the algorithm is designed. In the literature, a few authors have compared a specific algorithm with a specific network, but a means of comparing a match with an algorithm in general still needs to be found.

As a first step in such an analysis, a look at the various moves made by the algorithm is appropriate. By the very nature of the system, the data transfers required by

an algorithm in a SIMD machine take place according to some law. The controller will issue an instruction to each processor to move the data. This instruction is the same for each processor. Hence, the moves made by the processors are not random. The machine on which the algorithms have to run also has limited interconnection networks. These interconnections are not random, but follow a logical pattern to obtain most benefit from modular construction. In the interest of efficiency all the algorithms are written so that the moves conform with the hardware links on the interconnection networks.

The fact that the moves in an algorithm and the hardware links in an interconnection network match each other, means that the analysis of an algorithm can be accomplished by comparing how one interconnection network will simulate another one. Many periods are necessary to simulate one move of one network on another network.

I will use these specific simulations for several reasons. Each of the networks we will discuss has been proposed in some form in the literature and has been shown to be useful. There is very little in the literature directly comparing the simulation abilities of these networks for the purpose of analyzing algorithms. By using these simulations, the system designer and/or the algorithm designer may determine the communication time of the

algorithm and therefore be able to compare two algorithms on execution time rather than computation time. Since these networks have been shown to be useful, it is very likely that any network implemented may have to implement one of the other networks.

An alternative to this would be a dynamically re-configurable network, if such a network is available to execute the algorithm. Even then, the ability of the network to simulate other interconnections is very important to SIMD machine architects. First, to re-configure the network the machine will have to spend a few periods and then execute the algorithm. Second, the number of connections that can be implemented in the hardware is limited by cost and hardware complexity. Therefore, to decide on which network to implement, the system designer must consider the time it will take the network to simulate other interconnections which may be needed to perform various computational tasks.

As this analysis of how one network simulates another is carried out, I will evaluate the following measures of performance for the networks presented here. These performance measures are model independent and hence, will have to be adjusted so that the particular effects of the model at hand are included in a final analysis. This is a very simple step as will be shown by examples at the end of this presentation.

The technique that I am presenting can be applied to interconnection networks not specifically mentioned in this paper. The model independent nature of the results is significant because it means that the lower bounds determined using these techniques will be valid with respect to programming real SIMD machines to perform these simulations.

The performance measures that I will determine for the networks are:

(1) Diameter. The diameter is the max distance function that a data item has to travel in going from processor i to processor j , for all i and j on the interconnection network that is being analyzed.

(2) Bandwidth. The bandwidth is the number of data transfers possible without any conflicts in one message transfer stage. This measure is not entirely model independent, but can be adjusted to whatever model characteristic is available.

(3) Mean Distance between processors. This is the sum of distances from i to j , for any i and all j , divided by the number of processors in the network. (This measure is for the network that is available, not the one being simulated).

(4) Message Traffic. The message traffic is the ratio of the bandwidth and the system diameter. This

parameter gives the number of data items that can be transferred in one clock period.

I will first determine these measures to simulate one move of a network on another network. Then I will use these results to compare the execution of an algorithm on the various networks analyzed.

Simulations. Say that you have a ring network. As we have seen earlier, the ring network is defined as,

$$R(x) = (x+1) \bmod N$$

On this network we wish to simulate the move of a grid network. In the grid network, the data in one period can be moved a distance of $\pm n$ where $n = \sqrt{N}$. The ring network can simulate this move in a straightforward way in $n = \sqrt{N}$ steps. Thus

$$\text{Diameter } D_R^G = \sqrt{N}$$

To determine the bandwidth, first notice that every processor on the grid network will move its data a distance of \sqrt{N} in one step. Since all the data items are moving to an address $x + n$, this can be exactly duplicated by the ring network without any datapath conflicts. So,

$$\text{Bandwidth } B_R^G = N$$

The mean distance between processors on a two way ring network can be obtained by finding the distance from any one processor to all the other processors. This is a very straightforward calculation as follows:

$$\text{Mean Distance } MD_R = (0 + 2 \sum_{i=1}^{n/2-1} i + N/2) * 1/N$$

$$\begin{aligned} \text{For large } N \text{ this is } & (0 + \frac{2}{2} (N/2-1)(N/2-1 + 1) + \frac{N}{2}) * \frac{1}{N} \\ & = (\frac{N^2}{4} - \frac{N}{2} + \frac{N}{2}) * \frac{1}{N} = \frac{N}{4} \end{aligned}$$

So $MD_R = (N/4)$ and the

$$\text{Message Traffic } MT_R^G = B_R^G + D_R^G = N/\sqrt{N} = \sqrt{N}$$

Similar results can be obtained to simulate a move of the shuffle network on the ring network. On the shuffle network, the data in one period can be moved a distance of $(2*n) \bmod N$, where n is the processor's address. This is equal to a maximum distance of $N/2$. It will take the ring network at least $N/2$ steps to simulate this move. Thus,

$$\text{Diameter} = D_R^S = N/2$$

To simulate a move of the shuffle exchange network, the processors numbered 1 through $[N/2]$ will have to move their data forward by 1, while the processors numbered $[N/2]$ through $N-1$ will move their data backward by 1. The number of data items that can be transferred without any conflicts can be obtained as follows:

Data from processor $[N/2]$ moves to the right. In n steps, this data will have reached processor numbered $[N/2] + n$. Data from processors $N-1, N-2$, etc. moves left. In n steps, data from processor $N-n$ will have to processor $N-2n$. This can occur without conflict if $[N/2] + n$ and $N-2n$ differ at least 1. So,

$$[N/2] + n + 1 = N-2n$$

or

$$3n = N - [N/2] - 1$$

$$\text{or } 3n = [N/2] - 1 = [N/2]$$

$$\text{or } n = [N/6]$$

This shows that from the second half of the processors only $N/6$ of these can be allowed to move if there is to be no conflict. Thus of $N/2$ processors $N/3$ of these will have conflict. Of all the N processors, only $2N/3$ processors can be moved without conflict, or

$$\text{Bandwidth} = B_R^S = 2N/3$$

$$\text{An exact expression is } B_R^S = N - \sum_{i=1}^{\log_2 N} \frac{N}{4^i}$$

This can be obtained by fitting the number of conflicts to a geometric series. Thus the message traffic $MT_R^S = B_R^S + D_R^S = 2N/3 * 2/N = 4/3$.

Next, we simulate a move of the tree network on the ring network. A move on the tree network can move the data a max distance of $[(K-1)N/K]$. On the ring network this can be simulated in $[(K-1)N/K]$ steps, since the ring network can move the data a max of one processor at a time. So,

$$\text{Diameter} = D_R^T = [(\frac{K-1}{K})N]$$

The bandwidth that we have determined to simulate the other networks has been under the assumption that the network being simulated is able to move data, without conflict and in one time period, from each one of its processors to its logically adjacent processor. The tree cannot match this. On the tree network, if we ask each of the K sons to transfer data to its father, then we have a situation where data path conflicts can occur. On the other hand, if we

were to transfer data from father to son, then conflict free data transfers can occur. In this case the data being sent to the sons is not from different processors. Thus, if we restrict the network to conflict free and one to one transfers, then only N/K items can be transferred on a tree.

Of these $[N/K]$ data items, the ring network can transfer all the $[N/K]$ data items without conflict. This occurs since either all the data items are moving to a higher numbered processor (going from son to father) or all are moving to a lower processor (going from father to son). Thus,

$$\text{Bandwidth} = B_R^T = [N/K]$$

With these values for diameter and bandwidth obtained, the message traffic parameter can be calculated as,

$$MT_R^T = B_R^T + D_R^T = [N/K] + [(K-1)N/K] = 1/K-1$$

This time let us try with a grid network. As we have seen earlier, the grid network has been defined as:

$$G_1(x) = (x+1) \bmod N$$

$$G_n(x) = (x+n) \bmod N \quad n = \sqrt{N}$$

On this network we wish to first simulate the move of a ring network. In the ring network, data in one period can be moved a distance of ± 1 step. Since the grid network itself also has this capacity, it can simulate this move directly in one step. Thus,

$$\text{Diameter} = D_G^R = 1$$

The determination of bandwidth is a very straightforward process. This is so, because the two way ring network is a subset of the two way grid network. Thus all the data items that have to be moved on the ring can be moved on the grid also. Thus,

$$\text{Bandwidth } B_G^R = N$$

The mean distance on the grid network can be obtained as follows. First, we must obtain the distance to each processor in the row of the given processor. This will be,

$$\text{Row } (x) = 2 \sum_{i=1}^{n/2-1} i + n/2 \quad \text{for } n \text{ even} \quad n = \sqrt{N}$$

$$= 2 \sum_{i=1}^{n/2} i \quad \text{for } n \text{ odd}$$

and then the mean distance is given by

$$\begin{aligned} MD_G &= \frac{1}{N} \left[2 \sum_{i=1}^{N/2-1} (\text{Row } (x) + n*i) + \text{Row } (x) (n/2+1) \right] \quad \text{for } n \text{ even} \\ &= \frac{1}{N} \left[2 \sum_{i=1}^{[n/2]} (\text{Row } (x) + n*i) + \text{Row } (x) \right] \quad \text{for } n \text{ odd} \end{aligned}$$

The message traffic for this simulation will be

$$MT_G^R = B_G^R + D_G^R = N + 1 = \underline{N}$$

Similarly we will obtain results to simulate a move of the shuffle network on the grid network. On one move of the shuffle network, data can be moved a distance of $(2*x) \bmod N$, where x is the processor address. This is a maximum distance of $N/2$. On the grid network the data can be moved a distance of n or 1 . A combination of both the moves may be necessary. As an example, move a data item from row to row until it reaches the correct row. This requires moves

by n . Next move within the row until it reaches the correct processor. This requires moves by 1.

From this it is very clear that a max of $n/2$ moves from row to row and a max of $n/2$ moves in the row are necessary. Thus,

$$\text{Diameter} = D_G^S = \frac{n}{2} + \frac{n}{2} = n = \sqrt{N}$$

Determining by bandwidth is almost impossible, since simulating a move on the grid will depend on the method employed by the installation, (and there are many different possibilities). However, once the simulation method is established the bandwidth can be determined. If we choose the strategy mentioned earlier, then we can determine the bandwidth. The moves mentioned earlier resemble exactly the moves of the ring network with each column as a different ring. Thus, the number of conflict free moves will again be approximately $2N/3$. So

$$\text{Bandwidth } B_G^S = 2N/3$$

So, the message traffic is

$$MT_G^{SE} = B_G^S + D_G^S = 2/3 N + \sqrt{N} = 2/3 * N^{1/2}$$

Next we simulate a move of the tree network on the grid network. A move on the tree network will move the data a distance of $[(K-1)N/K]$. Since the grid network is capable of moving the information a distance where \sqrt{N} is

$$\sqrt{N} = \left[\frac{K^{i+1} - 1}{K - 1} \right]^{1/2} \approx K^{i/2}$$

So, the diameter is as follows.

$$\text{Diameter } D_G^T = \frac{(K-1)N}{K} * \frac{1}{K^{i/2}} = \left(\frac{K-1}{K}\right) \frac{K^{i+1}-1}{K-1} * \frac{1}{K^{i/2}} \\ \approx K^{i/2}$$

To determine the bandwidth, an argument similar to the one given in determining B_R^T can be followed. Doing this we find that at the most $[N/K]$ processors will be trying to move their data. If we restrict the movement of the data on the grid so that if the destination processor is at least N away, then the data moves up the column by \sqrt{N} and then moves by 1 along the row. With this plan the data can be moved with no datapath conflicts.

$$\text{The Bandwidth } B_G^T = [N/K]$$

With two parameters known the message traffic parameter can be calculated as follows:

$$\text{Message Traffic } MT_G^T = B_G^T \div D_G^T = N * \frac{K^{i/2}}{(K-1)N} * K$$

$$\text{Message Traffic } \approx \frac{K^{i/2}}{K-1} \approx K^{(i-2)/2}$$

This time let us consider the shuffle exchange network.

This network is defined as

$$S(P_{m-1}, P_{m-2}, \dots, P_1, P_0) = P_{m-2} \dots P_1 P_0 P_{m-1};$$

$$E(P_{M-1}, P_{M-2}, \dots, P_1, P_0) = P_{M-2} P_{M-1} \dots > P_1 P_0$$

On this network, we wish to first simulate the move of the ring network. In the ring network, data can be moved from processor address x to $(x + 1) \bmod N$. Such a move can alter all the bits in processor address x . To simulate this move the shuffle exchange network will alternately execute shuffles and exchanges until all the necessary bits

of the processor address have been altered. To address N processors $\log_2 N$ bits are necessary so that the diameter will be

$$\text{Diameter} = D_{SE}^R = 2 * \log_2 N$$

When the shuffle exchange network is trying to simulate the Ring network it has to move data items a distance of 1 processor away. This will not cause any conflicts during the shuffle move. During the exchange phase, because the move is the same for every processor, the bits affected occur in pairs and the exchange move always has an even number of items to be moved and placed in an even number of processors. Thus, both the shuffle and the exchange moves can be carried out without any data path conflicts. So, the bandwidth is,

$$\text{Bandwidth} = B_{SE}^R = N$$

The mean distance between processors on a shuffle exchange network can be obtained as follows. If the shuffle and the exchange are performed alternately, all the processors can be reached ultimately. When starting from processor x in a shuffle move, only processor $(2x) \bmod N$ can be reached. On the next exchange move two other processors can be reached. The next shuffle move can start from any one of the three processors reached and in this way can reach three others. On the exchange move starting from any of these three processors, three more can be reached and

continue on this way. Therefore, the mean distance between processors is,

$$MD = \frac{1*1 + 2*2 + 3*3 + 4*3 + 5*6 + 6*6 + 7*12 + 8*12, \dots}{N}$$

which is equal to $\log_2 N$

$$MD = \frac{\sum_{n=3}^N (1 + 4 + n-3) (3*2^{n-3} - 3(4*n-5))}{N}$$

This expression is approximately equal to $\log_2 N$.

The message traffic on the shuffle exchange network, when it is trying to simulate the ring network, is

$$\text{Message Traffic} = MT_{SE} = B_{SE}^R + D_{SE}^R = N/(2\log_2 N)$$

Similar results can be obtained for the grid network. In fact there is absolutely no difference between the results obtained for the ring and the results that can be obtained for the grid. This is due to the shuffle exchange network. On the shuffle exchange network, any data item can be moved from any processor to any other processor in a fixed time of $(2\log_2 N)$. Thus, the diameter of any move being simulated by a shuffle exchange network is $(2\log_2 N)$.

$$\text{Diameter} = \frac{\text{Any}}{D_{SE}} = 2 \log_2 N$$

When looking at the bandwidth, we find that any law which moves data items from processor x to $(x + a) \bmod N$ for all a can be simulated by the shuffle network any conflicts.

Thus, Bandwidth:

$$\text{Bandwidth} = \frac{\text{Any } (x \rightarrow x+a)}{B_{SE}} = \frac{N}{\text{Any } (x \rightarrow x+a)}$$

$$\text{and hence Message Traffic} = MT_{SE} = N/(2 \log_2 N)$$

In simulating the tree network on the shuffle exchange network, we have only to find the bandwidth since the

diameter is known. For this, an argument similar to the one given in determining B_R^T can be followed to find that at the most $[N/K]$ processors will be trying to move their data. If the data is to move from father to son, then the data follows the law $x \rightarrow x + a \bmod N$ and a is negative and does not have to be the same for each father. Then, since each father is moving data to only one of a group of sons no data path conflicts are found. Thus the bandwidth equals the number of data items moving.

$$\text{Bandwidth} = B_{SE}^T = [N/K]$$

Hence, the message traffic is

$$\begin{aligned} \text{Message Traffic} &= MT_{SE}^T = B_{SE}^T + D_{SE}^T = \frac{[N/K]}{2 \log_2 N} \\ &\approx \frac{N}{2K \log_2 N} \end{aligned}$$

Finally, we can obtain similar results on the tree network. To simulate a move of the ring network, the tree network has to move data from processor n to processor $n + 1$. To do this, the tree network will have to move at least one data item from one of its leaves through the root and back to the leaf on the other side of the tree. This will require $2i + 1$ steps. Hence the diameter is

$$\text{Diameter} = D_T^R = 2i + 1$$

If we restrict the tree network so that one son sends information to his father or the father sends information to one of his sons at a time, then we can move $[N/K]$ data items without any conflicts. Thus, the bandwidth to simulate a ring on the tree has an upper bound at $[N/K]$.

However, on trees of level three or more, additional conflicts are present. If the tree size is increased by 1 level then the number of conflicts are doubled. (See appendix for discussion). Therefore,

$$\text{Bandwidth} = B_T^R = [N/K] - 2^{i-3} - 1 = \frac{7N}{16}$$

$$\text{Message Traffic} = MT_T^R = B_T^R + D_T^R = \frac{7N}{16} + 2 \log_K N$$

$$\frac{7N}{32 \log_K N} \quad \text{for large } N.$$

When we simulate the move of the grid network on the tree network we have two different types of moves to simulate. Moving data from processor x to $(x+1) \bmod N$ is exactly the same as described above. The other move requires moving data a distance of \sqrt{N} away. This type of move produces \sqrt{N} conflicts. To resolve all these conflicts we need \sqrt{N} data transfer cycles. So in one data transfer cycle \sqrt{N}/K items can be made to reach their destination. (See Appendix for discussion). Therefore:

$$\text{Bandwidth} = B_T^G = \sqrt{N}/K$$

$$\text{Diameter } D_T^G = (2K \log_K N)$$

$$\text{Message Traffic} = MT_T^G = B_T^G \div D_T^G = (\sqrt{N}) / (2K \log_K N)$$

When we simulate the move of the shuffle network on the tree network we have to move data from processor x to processor $2x$. This means that at least one data item will have to move from a leaf through the root and down to another leaf. This, as we have seen earlier, would take $2i-1$ steps. As the number of levels on the tree increase, the number of data path conflicts also increase. Of the

N/K data items that we try to move each time only $(3N/4K)$ are able to reach their final destination without conflicts. (See Appendix for discussion). Therefore:

$$\text{Diameter} = D_T^{SE} = 2i-1 = 2\log_K N$$

$$\text{Bandwidth} = B_T^{SE} = (3N/8)$$

$$\text{Message Traffic} = MT_T^{SE} = B_T^{SE} + D_T^{SE} = (3N/16\log_K N)$$

TABLE 2

OF ON		RING	GRID	SHUFFLE EXCHANGE	TREE
	Diameter	1	\sqrt{N}	$N/2$	$\approx [(K-1)N/K]$
RING	Bandwidth	N	N	$\approx 2N/3$	$\approx [N/K]$
	Message Traffic	N	\sqrt{N}	$4/3$	$\approx 1/K-1$
GRID	Diameter	1	1	\sqrt{N}	$\approx K^{i/2}$
	Bandwidth	N	N	$\approx 2N/3$	$\approx K^{i-1} N/K$
	Message Traffic	N	N	$\approx 2/3 \sqrt{N}$	$\approx K^{(i/2)-1}$
S.E.	Diameter	$2 \log_2 N$	$2 \log_2 N$	1	$\approx 2i-1 \approx (2 \log_2 N)$
	Bandwidth	N	N	N	$\approx K^{i-1} \approx N/K$
	Message Traffic	$N/2 \log_2 N$	$N/2 \log_2 N$	N	$\approx K^i / (2i-1) \approx N/2K \log_2 N$
	Diameter	$2 \log_K N$	$2 \log_K N$	$2 \log_K N$	1
TREE	Bandwidth	$7N/16$	\sqrt{N}/K	$3N/8$	N/K
	Message Traffic	$7N/32 \log_K N$	$\sqrt{N}/2K \log_K N$	$3N/16 \log_K N$	N/K

PERFORMANCE MEASURES

CHAPTER 5

USING THE PERFORMANCE MEASURES

Now that we have these measures it is appropriate to see how they will help in matching an algorithm to a network. To do this I propose to examine three algorithms. One of these algorithms will require moves according to the laws of one of these networks. Another will require moves which do not match any of the networks that we have covered, and the third one will require data dependent moves.

The first algorithm that I chose is that of adding all the entries of a vector.

This algorithm does not match any of the networks discussed. The algorithm, as discussed earlier, can be written as follows:

- (1) Let $C = 1$
- (2) Add the data elements present in each processor
- (3) Move data from processor $(2i+C)$ to processor $2i$
- (4) If $C \geq N/2$ STOP
- (5) $C = 2 * C$
- (6) Go to step 2

We will determine the time taken to run this algorithm on each the networks when we have N processors and 2N data items. All the data items have been placed in the N processors previously and we are ready to execute the algorithm.

If the algorithm is to be run on a ring network;

(1) The first step will take only 1 step, since it is not part of the loop.

(2) Each time the loop is executed, step 2 is executed once on each processor. Since half of the elements are added each time the loop executes, the loop will have to execute $\log_2 N$ times. Thus, step 2 will be executed $\log_2 N$ times.

(3) This is the data communication step. Because it is part of the loop, it will be executed $(\log_2 N)$ times. However, each time this is executed, data will have to move a distance of $(1, 2, 4, \dots, C \text{ steps})$. Thus communication time for this algorithm is:

$$\sum_{i=0}^{\log_2 N} 2^i$$

(4) (5) These two steps are computation. Each takes one step. Since they are part of the loop, they will be executed $(\log_2 N)$ times each.

Thus, the total time to execute the algorithm on the ring network is communication time plus computation time, where:

Communication time = $\sum_{i=0}^{\log_2 N} 2^i$ and Computation time = $1 + 4 \log_2 N$.

To run the algorithm on a grid network, we will have to once again execute the loop $\log_2 N$ times. Next, steps (2, 4 and 5) will be executed $\log_2 N$ times each. Step 3, which is a communication step, will also be executed $\log_2 N$ times. Each time the communication step is executed, the data can be moved either North-South or East-West. As long as $C < \sqrt{N}$ the communication will have to take place according to the law $x \rightarrow x \pm 1$. After C has become $\geq \sqrt{N}$ the communication will be according to the law $x \rightarrow x \pm n$. In this way, the first half of the loop will be executed according to the first law and the second half of the loop will be executed according to the second law. Thus, the number of moves necessary for this step is:

$$\frac{1}{2}(\log_2 N) \sum_{i=0} 2^i$$

and the time for the whole algorithm is communication time plus computation time, where

Communication time = $2 \left[\sum_{i=0}^{\frac{1}{2}(\log_2 N)} 2^i \right]$ and Computation time = $1 + 4 \log_2 N$.

To run the algorithm as we have defined it on the tree network we would need to execute the loop $\log_2 N$ times. Steps (2, 4, 5, and 6) will be executed $\log_2 N$. Step 3, which is a communication step, will also be executed $\log_2 N$

times. On this network, each time the communication step executes, data has to be moved from the left half of the tree to the right half. Each move is similar to the move of the grid network and requires $2*i$ steps where i is the number of levels of the tree network. As we have seen earlier, when the tree is simulating a grid network, only $\sqrt{N}/2$ data items can be moved at a time. Each communication step will require $2*i*(2\sqrt{N})$ steps to execute.

The shuffle exchange network can finish each communication step in $2\log_2 N$ periods. The communication time for the whole algorithm is $2*\log_2 N*\log_2 N$.

Comparing the times on all four networks, (see Table 3B) we see that the computation time on all the networks is the same. The difference is the communication time. The ring network is the slowest. The grid and the shuffle exchange network compete for the fastest network. As long as the number of processors in the network is less than 16K, the grid network is faster. When there are more than 16K processors, then the shuffle network becomes faster. The results obtained are for the algorithm as it is written. If we are permitted to change the algorithm so that the transfer of data is from each son to his father (assuming two sons per father), then each communication step on the tree network would need two time periods. This

would make the total communication time $2 \cdot \log_2 N$ which is faster than on any other network.

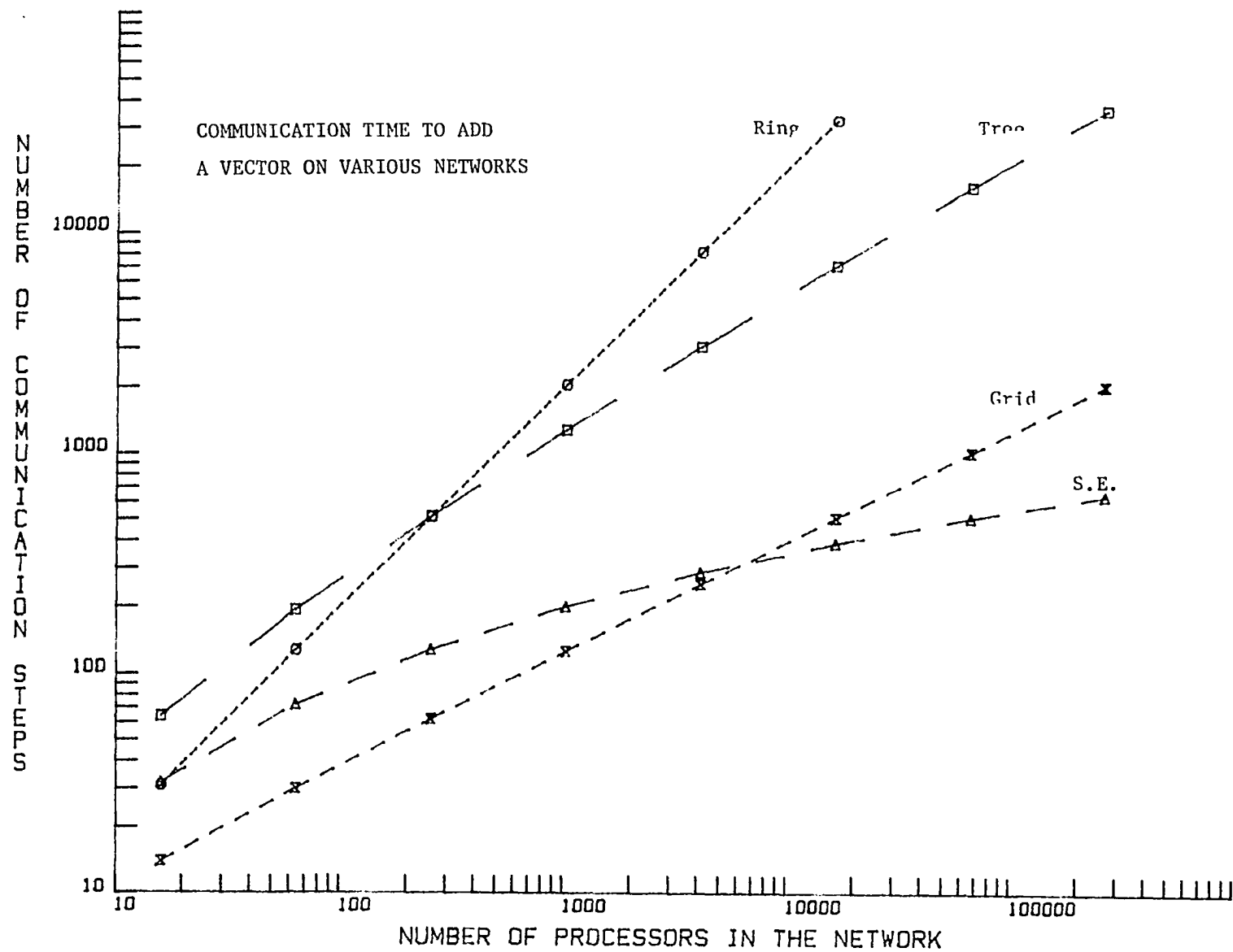
Network	Computation	Communication
Ring	$1 + 4 \log_2 N$	$\sum_{i=0}^{\log_2 N} 2^i$
S.E.	$1 + 4 \log_2 N$	$2 * (\log_2 N)^2$
Grid	$1 + 4 \log_2 N$	$\frac{1}{2} \log_2 N \sum_{i=0}^{\log_2 N} 2^i$
Tree	$1 + 4 \log_2 N$	$4\sqrt{N} \log_2 N$

TABLE 3A

As a second algorithm to analyze, I will obtain a product of two matrices. We will assume that both the matrices are square. We are using square matrices because modification is simple and it is easier to obtain the transposition of a square matrix on a shuffle network. We will assume that the two matrices conform with each other. We will start with the two matrices as shown in Table 4. Also assume that each processor has at least two free registers that can be used by the algorithm.

The algorithm is:

- (1) Transpose the second matrix.
- (2) Let $C = 0$.
- Loop (3) Multiply the two elements of the matrices in place.
- (4) Add all the elements of row i and store the



# of process commu.	RING NETWORK			GRID NETWORK			Computation Time
	commu./ compu.	commu./ total	commu.	commu./ compu.	commu./ total		
2 ⁴	31	1.824	0.646	14	0.824	0.452	17
2 ⁶	127	5.08	0.836	30	1.20	.545	25
2 ⁸	511	15.485	0.939	62	1.88	.653	33
2 ¹⁰	2047	49.93	0.980	126	3.07	.754	41
2 ¹²	8191	167.16	0.994	254	5.18	.838	49
2 ¹⁴	32767	574.86	0.998	510	8.95	.899	57
2 ¹⁶	131071	2016.48	0.999	1022	15.72	.940	65
2 ¹⁸	524287	7182.01	1.000	2046	28.03	.966	73
	<u>SHUFFLE EXCHANGE</u>				<u>TREE</u>		
2 ⁴	32	1.88	0.653	64	3.77	.790	17
2 ⁶	72	2.88	0.742	192	7.68	0.885	25
2 ⁸	128	3.88	0.795	512	15.52	0.939	33
2 ¹⁰	200	4.88	0.830	1280	31.22	0.969	41
2 ¹²	288	5.88	0.855	3072	62.69	0.984	49
2 ¹⁴	392	6.88	0.873	7168	125.75	0.992	57
2 ¹⁶	512	7.88	0.887	16384	252.06	0.996	65
2 ¹⁸	648	8.88	0.899	36864	504.99	0.998	73

TABLE 3B

Comparisons of various networks to add a vector for different numbers of processors in a network.

I	∅	a	1	b	2	c	3	d
	4	e	5	f	6	g	7	h
	8	i	9	j	10	k	11	l
	12	m	13	n	14	o	15	p
II	<u>∅</u>	<u>a</u>	1	e	2	i	3	m
	4	b	<u>5</u>	<u>f</u>	6	j	7	n
	8	c	9	g	<u>10</u>	k	11	o
	12	d	13	h	<u>14</u>	<u>l</u>	<u>15</u>	<u>p</u>
III	∅	b	<u>1</u>	<u>f</u>	2	j	3	n
	4	c	5	g	<u>6</u>	k	7	o
	8	d	9	h	10	l	<u>11</u>	<u>p</u>
	12	<u>a</u>	13	e	14	i	<u>15</u>	<u>m</u>
IV	∅	c	1	g	<u>2</u>	<u>k</u>	3	o
	4	d	5	h	6	l	<u>7</u>	<u>p</u>
	<u>8</u>	<u>a</u>	9	e	10	i	11	m
	12	<u>b</u>	<u>13</u>	<u>f</u>	14	j	15	n
V	∅	d	1	h	2	l	<u>3</u>	<u>p</u>
	<u>4</u>	<u>a</u>	5	e	6	i	7	m
	8	b	<u>9</u>	<u>f</u>	10	j	11	n
	12	c	13	g	<u>14</u>	<u>k</u>	15	o

Start algorithm with the two matrices in this position.

After transposition the matrices are in this position. Multiply and add. Underlined positions have final result.

Multiply and add. Underlined positions have final result.

Multiply and add. Underlined positions have final result.

Matrix product obtained in proper position.

TABLE 4

Algorithm setup to multiply two matrices.

result in a processor that is in $(C+1)$
column mod \sqrt{N} .

(5) Move the elements of the transposed matrix
to processor address $(x+\sqrt{N})$.

(6) $C = C + 1$.

(7) If $C < \sqrt{N}$ go to step 3, otherwise stop.

This algorithm has all the characteristics that we would expect to find in an algorithm. It runs in phases. Phase one requires you to transpose a matrix. Phase two requires multiplication of two matrices. Both the phases obey different laws for communication. The communication laws followed by both the phases of the algorithm are exactly the same as the laws of two of the networks that we have analyzed. Thus, we will be able to utilize the performance measures that we have developed.

Running the algorithm on the ring network:

(1) The first step, which requires transposing the second matrix, can be done on the shuffle exchange network without any conflict in a minimum of m shuffles. Thus, for this step the ring network has to simulate the shuffle network.

(2) Step three is a process of adding the contents of \sqrt{N} consecutive processors. This is the same as the previous algorithm analyzed.

(3) The next step is that of moving the elements of the transposed matrix by a distance of \sqrt{N} . This step can

be accomplished by the grid network in one step. Here we are simulating the grid network.

(4) The loop has to be executed \sqrt{N} times.

Time to run the algorithm on a ring network can be calculated as follows:

(1) To simulate a move of the shuffle network, the ring network needs $N/2$ steps. Since each time a shuffle has to be executed, the ring network can move $2N/3$ elements, the ring network will need:

$$m * N/2 * \lceil N/(2N/3) \rceil \text{ or } m * N \text{ steps.}$$

(2)(3) Step two and three on the ring will require one step each.

(4) Step four on the ring will require

$$\frac{1}{2} \log_2 N [4 \log(\sqrt{N}) + \sum_{i=0}^{1/2 \log_2 N} 2^i] \text{ steps.}$$

(5) This step is a simulation of the grid network on the ring network. Since the $D_R^G = \sqrt{N}$ and the bandwidth is N , this move can be accomplished in \sqrt{N} steps.

(6)(7) These two steps can be done in one step.

$$\text{Computation time} = 1 + (1 + 4 \log_2 \sqrt{N} + 2) * \sqrt{N}$$

$$\text{Communication time} = (\log_2 N) * N + \left[\sum_{i=0}^{1/2 \log_2 N} 2^i + \sqrt{N} \right] * \sqrt{N}$$

Running this algorithm on a shuffle exchange network, the first step being a shuffle step, this one move can be executed in m steps (Stone (4)). The next communication

step is the same as that on the previous algorithm. The final communication step will need $2m$ steps since the diameter of the grid network on the shuffle exchange network is $2m$ and the bandwidth is N . This results in the following:

$$\text{Computation time} = 1 + (1 + 4\log_2 \sqrt{N} + 2) * \sqrt{N}$$

$$\text{Communication time} = \log_2 N + [2 * (\log_2 N)^2 + 2 \log_2 \sqrt{N}] * \sqrt{N}$$

The next step is to run this algorithm on the grid network. To do this, we need to simulate the shuffle move first. As seen previously, the diameter of a shuffle network on a grid is \sqrt{N} and the bandwidth is $\frac{2N}{3}$. The m shuffle moves will need,

$$m * \sqrt{N} * \lceil N / (2N/3) \rceil = 2m \sqrt{N} \text{ steps.}$$

The next communication move is similar to the algorithm already examined. The time is according to that previously obtained. The next communication step is to shift data \sqrt{N} distance away. This can be done on the grid network in one step. Hence, the time to execute the algorithm on the grid network can be written as,

$$\text{Computation time} = 1 + (1 + 4\log_2 \sqrt{N} + 2) * \sqrt{N}$$

$$\text{Communication time} = 2(\log_2 N) \sqrt{N} + (2 + \sum_{i=0}^{1/4 \log_2 N} 2^i + 1) * \sqrt{N}$$

Finally when running this algorithm on the tree network, the shuffle move will require

$$m * N + \frac{3N}{16 \log_2 N} = \frac{16}{3} (\log_2 N)^2$$

The next communication move is similar to the algorithm already examined and the times for step four are already known. Step five requires simulating the move of the grid network. Thus time required to execute step five is

$$\sqrt{N} * [N + (2K \log_2 N)] = \sqrt{N} * (4\sqrt{N} \log_2 N) = 4N \log_2 N$$

Thus the Computation time is $1 + (1 + 4 \log_2 \sqrt{N} + 2) * \sqrt{N}$ and the Communication time is $\frac{16}{3} (\log_2 N)^2 + 4N \log_2 N + 4N \log_2 N$

Once again, for this algorithm, we obtain similar results. The computation time stays constant as long as the algorithm is not changed. The execution times follow similar patterns as before. The ring network is the slowest network when the number of processors is large. Notice, however, that the tree network is the slowest to begin with. This is seen initially because, during the step of adding the products of each row, the ring network behaves as if it is made up of \sqrt{N} chains \sqrt{N} processors long and this reduces its diameter. However, as the number of processors gets larger, even this help from the algorithm is not enough. This shows that the ring network

can execute algorithms faster than the tree network.

Communication time to multiply two matrices.

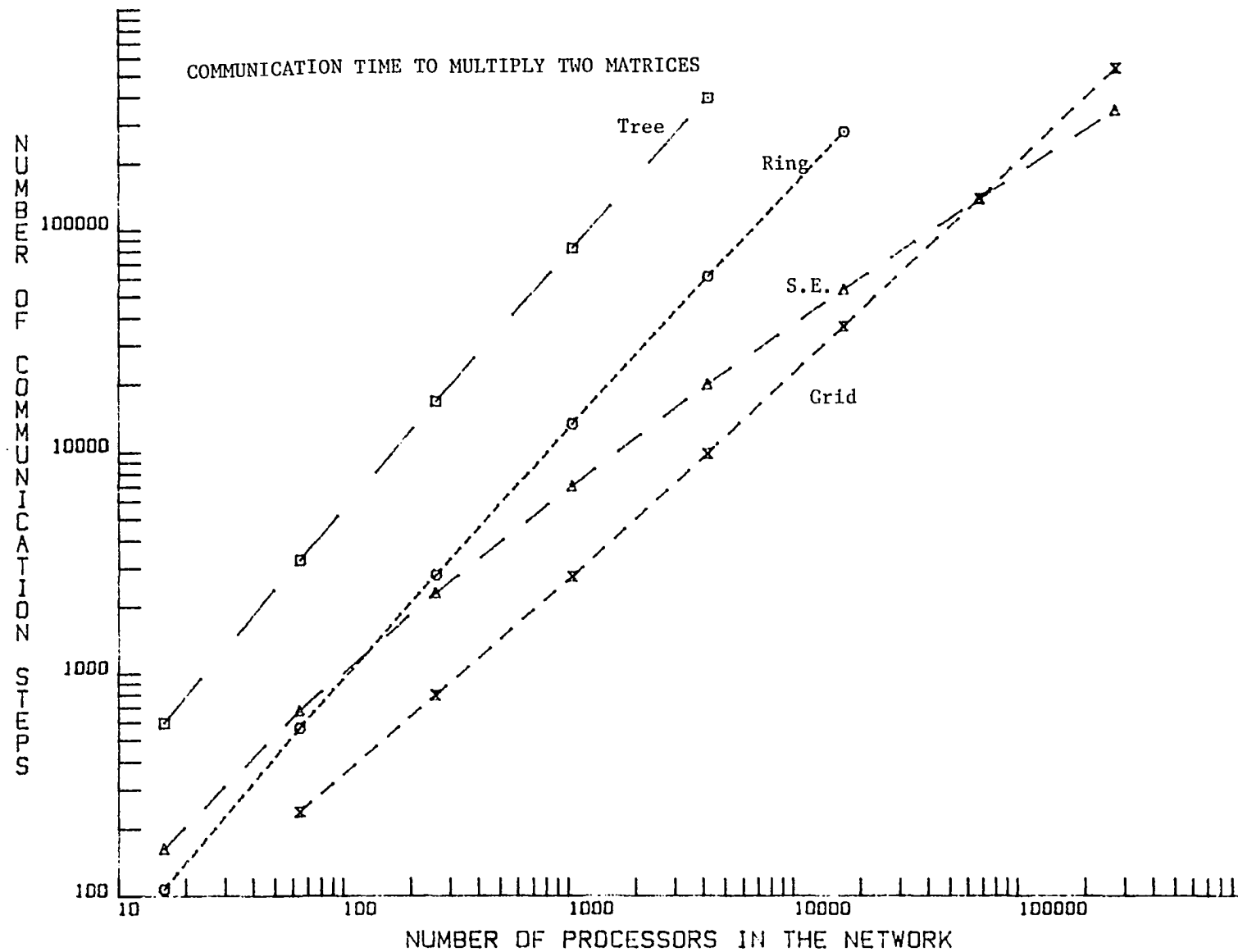
Network	Computation	Communication
Ring	$1 + (3 + 4 \log_2 \sqrt{N}) * \sqrt{N}$	$\frac{1}{2} \log_2 N$ $N \log_2 N + [\sum_{i=0}^{1/2 \log_2 N} 2^i + \sqrt{N}] \sqrt{N}$
S.E.	$1 + (3 + 4 \log_2 \sqrt{N}) * \sqrt{N}$	$\log_2 N + (2 * (\log_2 N^2 + 2 \log_2 \sqrt{N}) \sqrt{N})$
Grid	$1 + (3 + 4 \log_2 \sqrt{N}) * \sqrt{N}$	$.2 \sqrt{N} \log_2 N + (3 + \sum_{i=0}^{1/4 \log_2 N} 2^i) * \sqrt{N}$
Tree	$1 + (3 + 4 \log_2 \sqrt{N}) * \sqrt{N}$	$\frac{16}{3} (\log_2 N)^2 + 8 N \log_2 N$

TABLE 5

Also notice that the addition of all the elements of a row on the ring and the grid network require the same time. This is because the grid can use only ± 1 function for this addition. This is how the ring network is defined. Hence, for this step of the algorithm it makes no difference whether the ring or the grid network is used.

As the number of processors becomes larger, the shuffle exchange network once again becomes the most efficient network. This is not true when the number of processors is small.

The third algorithm that I will analyze is a sorting algorithm. This algorithm was originally introduced by Batcher. It sorts by merging. Since this is a sorting algorithm, it will test our networks to their fullest. A sorting algorithm must be able to move data anywhere in the network. The final goal of this algorithm is the same as



# of processes	RING NETWORK			GRID NETWORK		Computation time	
	comm. comm.	comm./ comp.	comm./ total	commu.	comm./ comp.		comm./ total
2^4	108	2.400	.706	72	1.600	0.615	45
2^6	568	4.694	.824	240	1.983	0.665	121
2^8	2800	9.180	.902	800	2.623	0.724	305
2^{10}	13280	18.019	.947	2752	3.734	0.789	737
2^{12}	61376	35.498	.973	9856	5.700	0.851	1729
2^{14}	278400	70.144	.989	36608	9.223	0.902	3969
2^{16}	1244928	138.927	.993	139776	15.584	0.940	8961
2^{18}	5504512	275.652	.996	543744	27.229	0.965	19969
	SHUFFLE EXCHANGE				TREE		
2^4	164	3.644	.785	597	13.267	0.930	45
2^6	678	5.603	.849	3264	26.975	0.964	121
2^8	2312	7.580	.883	16725	54.836	0.982	305
2^{10}	7050	9.566	.905	82453	111.877	0.991	737
2^{12}	19980	11.556	.920	393984	227.868	0.996	1729
2^{14}	53774	13.549	.9311	836053	462.598	0.998	3969
2^{16}	139280	15.543	.9408	389973	936.276	0.999	8961
2^{18}	350226	17.538	.946	3775064	1890.453	1.000	19969

TABLE 6

Comparisons of various networks to multiply two matrices for different numbers of processors in a network.

that of any other sorting algorithm: arrange the data in such a way that the smallest key is located in processor numbered 0 and so on. To do this, the algorithm requires a compare and exchange step. This step is executed as follows.

First, move the data to be compared from a higher numbered processor to a lower numbered processor. Then execute the compare step in the lower numbered processor. Leave the smaller valued data there and move the higher valued data back to the higher numbered processor. These moves from high numbered processor and back to the higher number processor make up the communication times.

The distance the data has to be moved for each comparison is not a fixed value. It is dependent on the index value of the outer and the inner loops. There is also a third constant that is used as a mask to inhibit some of the processors from sending out their data. In general we can say that for every execution of the inner loop, $N/2$ data items have to travel from a higher numbered processor to a lower numbered processor and the same number of data items have to return. The communication time required for data items to reach the designated processor for comparison is the same as the time required to return after the comparisons are made. The algorithm:

(1) Set $t = \lceil \log_2 N \rceil - 1$ and $p = 2^{t-1}$

- (2) Set $q = 2^{t-1}$, $r = 0$, $d = p$
- (3) For all i such that $0 < i < N-d$ and $(i \text{ and } p) = r$, step four.
- (4) Move data item from processor $(i + d + 1)$ to processor $(i + 1)$
- (5) Compare the two data items in processors.
- (6) Return the high valued data to processor $(i + d + 1)$.
- (7) If $q = p$, then go to 9.
- (8) Set $d \leftarrow q-p$ then $q = q/2$ $r = p$, go to 4
- (9) Set $p = \lceil p/2 \rceil$
- (10) If $p \geq 2$ go to 2
- (11) Stop.

In this algorithm the communication takes place in step 4 and step 6. The comparison in step 5 and the calculations in the rest of the steps are book-keeping. The book-keeping and the comparison times are independent of the network: they require the same number of periods on all the networks. We will analyze step 4 for each network. This will give us the information we are looking for.

The ring network. Since in one period this network can move the data a distance of 1 unit, the number of periods required for step 4 on this network is

$$\sum_{n=1}^{\lceil \log_2 N \rceil - 1} \left[\sum_{m=0}^n 2^{n-2m} \right].$$

This can be written in a form easier to calculate.

$$\sum_{n=1}^{\lceil \log_2 N \rceil - 1} \left(\left[\sum_{m=0}^n \frac{(2^m - 1)N}{2^n} \right] + \frac{N}{2^n} \right).$$

Thus, the communication time on the ring network is twice the above expression. The computation time, which is the same for all networks, is given by the following expression.

Step 1 --> 1

Step 2 --> 1

Step 3 --> 3 * [log₂ N + 1]

Step 5 --> 1 * [log₂ N][log₂ N + 1] * 1/2

Step 7, 8 --> 4 * [log₂ N][log₂ N + 1] * 1/2

Step 9, 10 --> 2 * [log₂ N + 1]

Total computation time on all networks is,

$$2 + [\log_2 N + 1] [3 + 1/2 \log_2 N + 2 * \log_2 N + 2]$$

$$2 + [\log_2 N + 1] [5 + 2.5 \log_2 N]$$

On the shuffle exchange network each time step 4 is executed, irrespective of the distance the data has to move, we will need 2 [log₂ N] communication steps. Step 4 is executed [log₂ N] * [log₂ N + 1] times.

The communication time on the shuffle exchange network is given by twice the following expression.

$$2[\log_2 N] * [\log_2 N] * [\log_2 N + 1]$$

$$2[\log_2 N]^2 * [\log_2 N + 1]$$

Executing the same algorithm on the grid network, we need to look at the distance the items have to be moved. Some

of these moves can be made by moving the data a distance of \sqrt{N} each time, while the others require a move of a distance of one. The distance the various data items have to move was seen earlier. In that expression, as long as $2^{n/j}$ is less than \sqrt{N} the moves can be made along \sqrt{N} distances. For all values of $2^{n/j}$ greater than \sqrt{N} , the moves have to be made by moving the data items one unit away, each time. Thus, the time to execute the communication step can be written as,

$$\sum_{i=1}^{\log_2 \sqrt{N}} ([\sum_{j=0}^i (2^{j-1}) \sqrt{N}/2^i] + \sqrt{N}/2^i) +$$

$$\sum_{i=\log \sqrt{N}+1}^{\log N-1} [\sum_{j=i-\log \sqrt{N}+1}^{\log N-1} N/2^i] +$$

$$\sum_{i=1+\log \sqrt{N}}^{\log N-1} [(\sum_{j=0}^{i-\log \sqrt{N}} (2^{j-1}) N/2^i) + N/2^i].$$

In this expression, the first two terms account for all the moves that can be made by moving the data items a distance on \sqrt{N} on each move. The remaining moves, which have to be made by moving the data a distance of 1 each time, are accounted for in the third term. The above expression accounts for the time required to execute step 4 in the algorithm. Step 6 has a similar expression. Thus, the communication time to execute the entire algorithm is twice that of the above expression.

Performing a similar analysis on the last network that I am using, we find each communication step requires data to be moved across the root of the tree. Data will move from the left half to the right half during step 6 and from the right half to the left half during step 4. There is a very good possibility of data path conflicts occurring here. Also, the moves are not according to any fixed pattern that can be analyzed on the tree network. Some moves can be made in $i + 1$ steps, while others require $2i + 1$ steps. To complete this analysis, I will choose the average value of steps required for each move. The average value of steps that each move will take is $(3i)/2 + 1$ steps. Next, to account for the data path conflicts, we need to multiply by $N/2$ because only $N/2$ processors are sending out the data. We then divide it by the bandwidth, which is approximately $N/5$. Each communication step will require $\lceil 5/2 \rceil * \lceil (3i)/2 + 1 \rceil$ steps.

Since step 4 is executed $(1/2)\log_2 N * (\log_2 N + 1)$ times, step 4 and step 6 each will require,
 $3 * \lceil (3/2)\log_2 N + 1 \rceil * (1/2)\log_2 N * \lceil \log_2 N + 1 \rceil$ time to execute.

Network	Communication time
Ring	$\sum_{i=1}^{\lceil \log_2 N - 1 \rceil} \left[\left(\sum_{j=0}^i (2^j - 1) N / 2^i \right) + N / 2^i \right]$
Shuffle	$[\log_2 N]^2 * [\log_2 N + 1]$
Tree	$(3/2) * ([(3/2) \log_2 N + 1]) * \log_2 N * [\log_2 N + 1]$
Grid	$\sum_{i=1}^{\log_2 \sqrt{N}} \left(\left[\sum_{j=0}^i (2^j - 1) \sqrt{N} / 2^i \right] + \sqrt{N} / 2^i \right) +$ $\sum_{i=\lceil \log_2 \sqrt{N} + 1 \rceil}^{\lceil \log_2 N - 1 \rceil} \left(\left[\sum_{j=i - \log_2 \sqrt{N} + 1}^i (2^j - 1) \sqrt{N} / 2^i \right] + \right.$ $\left. \left[\sum_{j=0}^{i - \log_2 \sqrt{N}} (2^j - 1) N / 2^i + N / 2^i \right] \right)$

TABLE 7

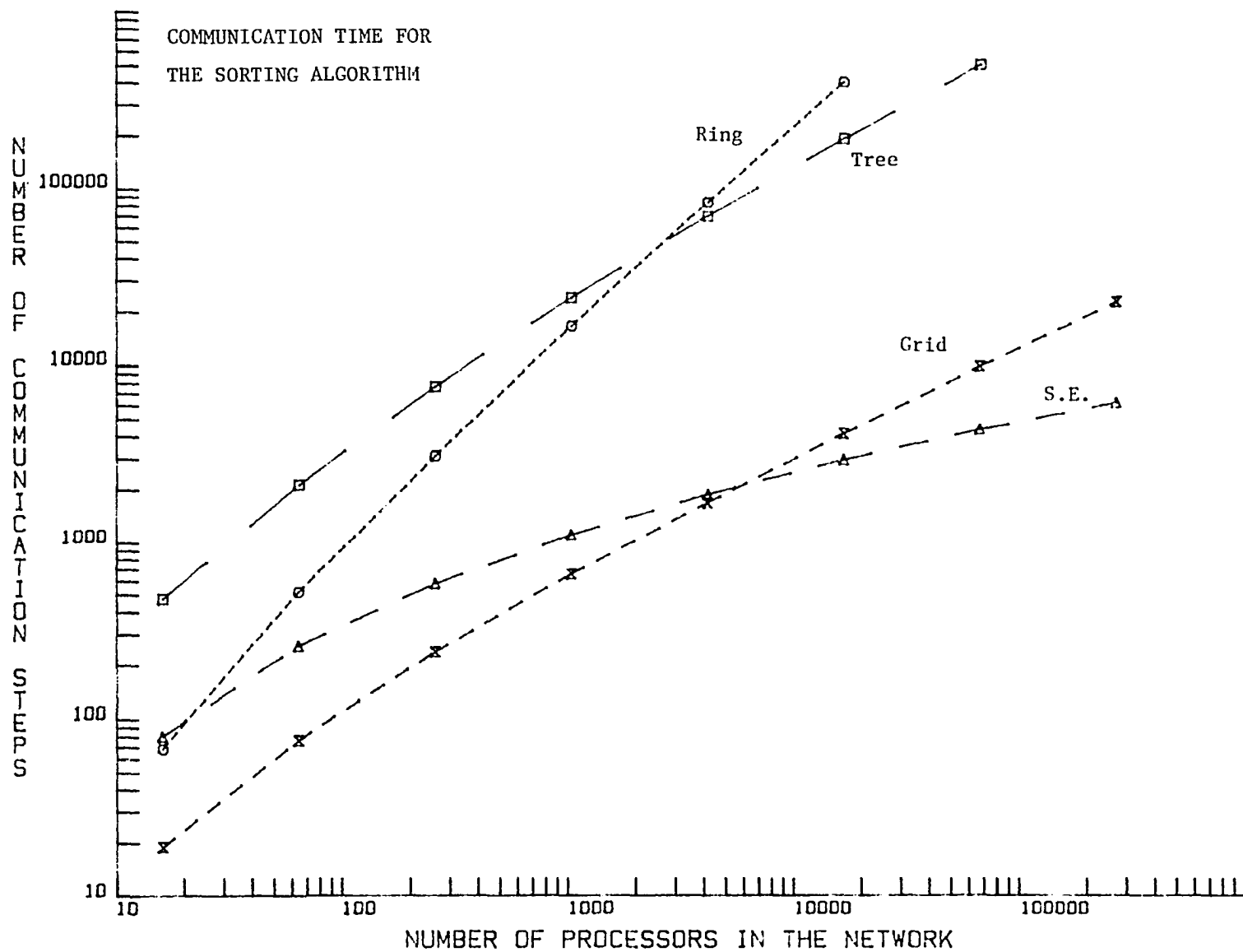
The computation time on all networks is,

$$5 * \log_2 N * [1 + 1/2 + (1/2)(\log_2 N + 1)] + 2$$

From the three algorithms analyzed, a few patterns seem to emerge. A careful study shows that:

(1) Communication between processors requires the least amount of time if the move required by the algorithm exactly matches the move that can be executed by the network in one step. The reasoning is obvious. Unfortunately, during the life of a network, it will be asked to simulate moves not native to it. Some degradation is to be expected on almost all algorithms.

(2) Communication between processors begins to degrade when the network has to simulate a move that is foreign to it. The amount of degradation varies from



# of processors	SHUFFLE EXCHANGE			TREE		Computation	
	Comm.	comm./ comp.	comp./ total	comm.	comm./ comp.	comm./ total	Time
2^4	80	0.975	0.494	469	5.720	0.851	82
2^6	256	1.658	0.624	2112	13.895	0.933	152
2^8	576	2.380	0.704	7509	31.030	0.969	242
2^{10}	1100	3.125	0.758	23573	66.969	0.985	352
2^{12}	1872	3.884	0.795	68352	141.809	0.993	482
2^{14}	2940	4.652	0.823	187413	269.540	0.997	632
2^{16}	4352	5.426	0.844	492885	614.570	0.998	802
2^{18}	6156	6.206	0.861	1255104	1265.276	0.999	992
		<u>GRID</u>			<u>RING</u>		
2^4	19	0.232	0.188	68	0.829	0.453	82
2^6	76	0.500	0.333	516	3.395	0.772	152
2^8	237	0.979	0.495	3076	12.711	0.927	242
2^{10}	654	1.858	0.650	16388	46.557	0.979	352
2^{12}	1679	3.483	0.777	81924	169.967	0.994	482
2^{14}	4112	6.506	0.867	393220	662.184	0.998	632
2^{16}	9745	12.151	0.924	1064964	1327.885	0.999	802
2^{18}	22546	22.728	0.958	2056756	2526.972	1.000	992

TABLE 8

Communication time for the Sorting Algorithm

network to network and it is a function of (a) the network itself, (b) the number of times the simulation has to be performed, and (c) the diameter and the bandwidth of the move being simulated.

From the analysis of the algorithms, we can see that the communication time depends on the diameter and the bandwidth. If the diameter of the move being simulated is small, then the message can reach its destination in fewer steps. The diameter on a network can be reduced by increasing the number of processors any one processor is connected to, or by switching the interconnecting structure on the processors so that it matches the move required by the algorithm. Increasing the number of processors any one processor is connected to has some limitations. First, it is very expensive and next, there are physical limitations. Each processor has a fixed number of I/O ports available. Switching the interconnection structure so that the network matches the move on the algorithm is being examined by researchers. A machine capable of this will require special hardware. Even with switching hardware, the network will have its limitations as to how many different interconnections are possible. In addition, the time used to switch hardware from one type of network to another and then back will have to be accounted for and included in the communication time. This time will have to include the time required for any changes necessary in the processor

memory for various tables. Thus, the engineer will have to decide if switching the network is time saving, or time consuming. Another factor that will be affected is the bandwidth. On any move, a large bandwidth is desired since this would indicate that many data items will reach their final destinations without conflicts. Increasing the bandwidth can be done the same way that the diameter is reduced. However, extra care should be exercised because increasing the number of interconnections could also decrease the bandwidth by introducing extra data path conflicts.

The effect of the diameter and the bandwidth can be studied simultaneously by the use of the message traffic parameter. This parameter was earlier defined as the ratio of the bandwidth to the diameter. Thus, either increasing the bandwidth or decreasing the diameter will increase the message traffic parameter. A large message traffic parameter is desired. The limits on the message traffic parameter are from N to $1/N$. The message traffic parameter of N indicates that all the N processors can transfer data in one step and an exact match between the network and the move required by the algorithm is achieved.

(3) As the number of processors in the interconnection network increases, the communication time for each message traffic stage is also increased. These two increases do not have a linear relationship. This can

be seen by comparing the execution times of the algorithms analyzed. For values of N that are small, we see that the grid network takes less time to execute than the shuffle exchange. As the number of processors in the network increase the grid network requires more time to finish execution. At some point the shuffle exchange network becomes better than the grid network. A similar result can be seen between the tree network and three ring network. This effect can be summarized as follows.

(4) A network with a smaller average distance between processors will be the more efficient network of the two as the number of processors in the network becomes very large.

For the four networks analyzed the average distances between processors are:

Ring	$O(N/2)$
Tree	$O(N^{1/2})$
Grid	$O(N^{1/2})$
S.E.	$O(\log_2 N)$

Average distance between processors on various networks

TABLE 9

Looking at the three algorithms analyzed, we see that for a large value of N , the ring network needs the most time and the shuffle exchange network needs the least time. The grid network and the tree network fall in between the ring and the shuffle exchange network as Table 9 indicates. The average distance between the processors is a parameter that

is similar to the network diameter. This parameter can be altered just as the network diameter is altered. If the number of processors each processor is connected to is increased, then both the network diameter and the average distance between processors is reduced.

The number of processors each processor in a grid network is connected to is $4(1-1/\sqrt{N})$ and each processor in the shuffle exchange network is connected to $3-2/N$. Even though each processor in the grid network is connected to more processors than each processor in the shuffle exchange network, the average distance between processors on a shuffle exchange network is less than that on the grid network. This shows that the interconnection geometry is a factor in the communication time. Indiscriminately adding interconnections to reduce the diameter in hope of reducing communication time is not the answer.

If the number of processors is not arbitrarily large, then a different effect is noticed from the graphs shown. In the region from about 2^8 - 2^{12} processors in the network, the grid network is the more efficient. Then comes the shuffle exchange followed by the tree, and the ring is last, being the slowest. In this region the geometry of the network is not the dominating factor. In this region, the number of processors each processor is connected to dominates the communication time. As a result, the grid network is the most efficient of the four.

CHAPTER 6

CONCLUSIONS

The rapid reduction in the cost of microprocessors and the advances in VLSI technology have made it feasible for us to have a computer with 2^{10} to 2^{16} processing elements. In order to design such a system, one capable of operating as a SIMD machine, an analysis of the interconnection network is necessary.

The analysis presented in this paper is intentionally system independent and hence, may be generalized and used to compare other possible networks. The direction taken in this article is from the network point of view. Many other authors have done analysis as far as communication time is concerned. Their major thrust has been analyzing different algorithms and modifying one of them so that it executes most efficiently on a given network. That is the thrust so far has been from the algorithm point of view. What this analysis shows is:

(1) Communication of data on a single instruction, multiple data machine is a very important factor which must

be considered when the machine is being designed and when algorithms are being executed on it.

(2) The network selected for the machine interconnection structure will be asked to simulate other networks in its lifetime. How well this network simulates other networks will determine how efficient this machine is under different operating conditions.

(3) I have shown the factors that affect the simulations and the parameters that measure the match between the algorithm and the network on which the algorithm has to execute.

(4) No single network can be shown to be the best network for a SIMD machine but some networks are better than others. (a) A superset network is at least as efficient as the subset network, most of the times more efficient. (b) A network with smaller average distance between processors is better than a network with a larger average distance between processors when the number of data items to be moved during a communication step is large. (c) A network which has each of its processors connected to more processors than another will be the more efficient network when the number of data items involved in a communication step is small. The analysis shows that even under the limited constraint of the least execution time, the various networks lead or fall back as the number of processors in the network varies.

(5) I have shown that in the SIMD machine, the time to execute the algorithm depends on: (a) the average distance between processors, when the number of processors in the network is large, and (b) the average number of processors a given processor is connected to when the number of processors in a network is not so large.

The break even point between two networks is not clearly defined. It depends on the networks being considered and the algorithm being executed. As a result of this, superset networks are being considered by many designers.

(6) A network that is the superset of another network will require less execution time, irrespective of how many processors are in the network. This can be seen by comparing the ring and the grid networks. The ring network is the subset of the grid network. Each processor in the superset network will always be connected to more processors and hence, will be more efficient when the number of processor is not large. The superset network will also have a smaller average distance between processors and therefore will be more efficient when the number of processors in a network is large.

The forming of a superset of a network can be carried out by increasing the number of processors that are connected to anyone processor. This has the effect of reducing the diameter and the average distance between

processors. This could possibly increase the bandwidth. All these factors point towards reducing the execution time of an algorithm on a superset network. Unfortunately, the forming of a superset network increases the cost of the network.

When examining the shuffle exchange network, we find that this network can simulate any move, of any algorithm, in a fixed number of steps. This gives us the upper bound on the execution time of an algorithm. A given network should require less time, otherwise the shuffle network should be used. The shuffle network for smaller values of N is not as efficient as the grid network. The break even point between these two networks is not a fixed value of N .

If a systems architect is concerned with minimizing hardware cost, but is willing to use $2\log_2 N$ moves for any simulation, then the shuffle exchange network would be a good choice. If execution speed is the architect's chief concern, then some superset of the shuffle exchange network would be a good choice.

A superset of the shuffle exchange network can be formed by connecting each processor x to $x * 2^i \bmod N$, where i can take on different values. For $i = 1$ we obtain the shuffle exchange network. A hybrid superset network can be formed by connecting the processors both as a grid, and as a shuffle exchange network.

Judging from the algorithms analyzed and the conclusions drawn, such a hybrid network would be an excellent choice. As long as the number of processors in a network is small, then the grid connection would provide excellent speed and the shuffle exchange network would provide the upper limit as the number of processors in the network increase. This type of hybrid network would be a very flexible network and have a good computational speed. The cost of the interconnections would still be a relatively small portion of the total cost of the system.

The results of this study provide a means by which algorithms and networks for SIMD machines can be compared. The comparisons made will aid both the system designer and the algorithm author. The system engineer will be able to compare between various networks and evaluate their simulation abilities. The algorithm author will design his algorithms so that data transfer can be easily accomplished.

Currently more and more super computers based on SIMD machines are being proposed. The study, simulation and construction of different designs will require comparisons between them. This study provides one means to achieve this.

OTHER RESEARCH DIRECTIONS

This article has shown that different networks are more efficient under different conditions. Also a super set network is more efficient than a subset network. Future research can be done to determine how to construct the super set networks. Some possible ideas would be:

(1) It is obvious that a fully connected network is the most efficient of all networks. The cost of fully connecting a network of 2^{12} to 2^{14} processes is probatively large. Everyone today is resigned to a subset of a fully connected network. A subset of a fully connected network could be nodes of say 4 processors each with each node connected to another and the four processors in the node being fully connected. How should the interconnections between the nodes be made? With todays technology it is possible to place these four fully connected processors on a single chip. How to connect these chips? Which pins must be brought out of these chips?

(2) In the paper we have seen that a super set network is more efficient than a subset network. How should this super set network be constructed? For example,

the S.E. network can be defined as $x \rightarrow x \cdot 2^1$. If instead of this connection a net with $x \rightarrow x \cdot 2^j$ with j having two or three or possibly more values we will have a super set network. Of all the possible choices of j some will have a definite advantage over others. Which values of j provide the most speed up of communication time?

(3) Another research direction would be in the direction of re-configurable networks. Which networks should be used? How does the time to re-configure a network effect the communication time? Since each processor has a limited number of ports, are these ports effectively utilized?

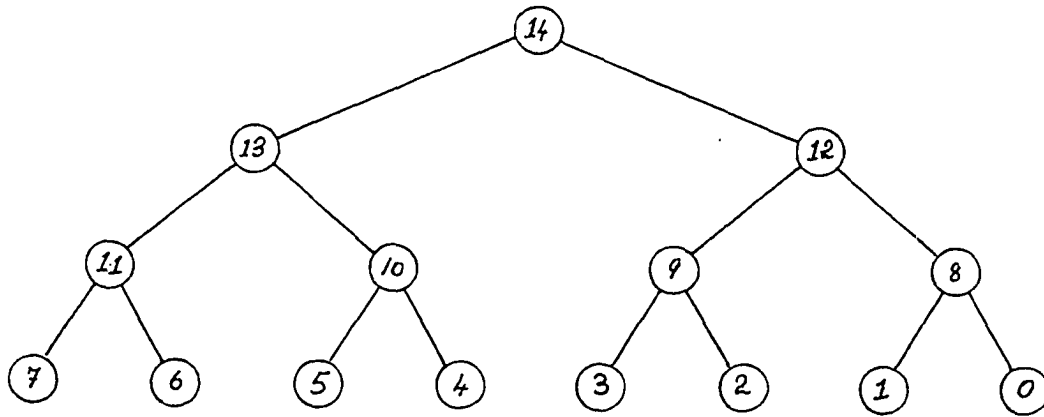
APPENDIX I

To move the data on a tree network from processor x to processor $(x + A) \text{ Mod } N$, we will use the following scheme. As we have seen earlier, each time a move is made, we will move N/K data items at a time. When these N/K data items have reached their destination, another batch of N/K items will be started. The N/K data items that have to be moved will not be selected at random. To select N/K data items, we select any one of the K sons which have a common father. Call this son x . Next, we go over to an adjacent father and select a son that is numbered $(K + x + 1) \text{ mod } K$. This son will be from a father with a higher number. If we go to a father that has a lower number, then, the son selected will be numbered $(x - K - 1) \text{ mod } K$. This way when a son is selected from each father, we will have selected N/K sons. These N/K sons will be allowed to transfer data. For the next N/K data item moves, the first son we select will be numbered $(x + 1) \text{ mod } K$ and the rest of the sons as described above. For such a selection scheme, the data movements and the conflicts that occur follow a pattern and can be studied.

To see how and where conflicts occur, I will examine two trees. One of level 3 and one of level 4 will be examined. All the trees will have two sons per father, since in each of the algorithms, I have used tree networks that have two sons per father.

When $a=1$, simulate the ring network.

Level 3 tree



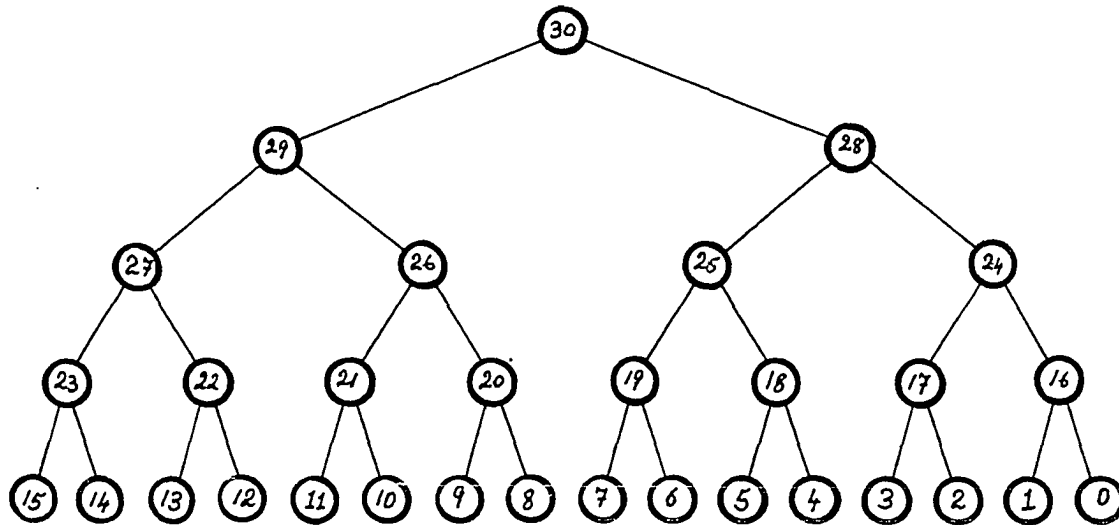
From this tree, in accordance with the selection scheme described, we can choose either 0,3,4,7,8,11,12, or 1,2,5,6,9,10,13,14, as a group of sons to transfer data.

Thus the datapaths will be:

0 - 8 - 1	1 - 8 - 12 - 9 - 2
3 - 9 - 12 - 14 - 13 - 10 - 4	2 - 9 - 3
4 - 10 - 5	5 - 10 - 13 - 11 - 6
7 - 11 - 13 - 14 - 12 - 8	6 - 11 - 7
8 - 12 - 9	9 - 12 - 14 - 13 - 10
11 - 13 - 14 - 12	10 - 13 - 11
12 - 14 - 13	13 - 14

In the first group, if we block data item number 7, and in the second group, if we block data item number 9 or 14, then all the other data items can reach their destinations without conflicts. We have one conflict for each group of $N/2$ data items for a level 3 tree.

In a level 4 tree;

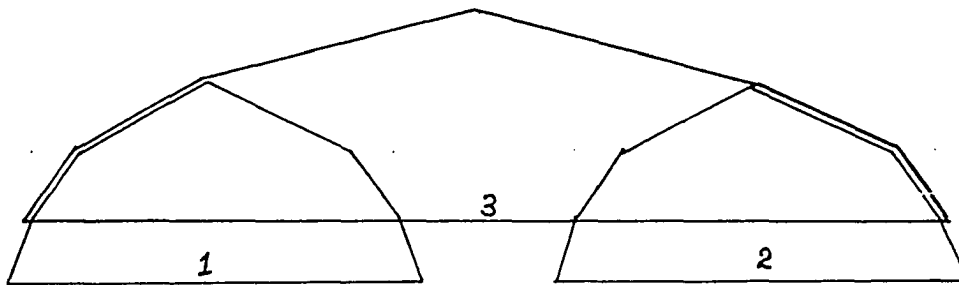


From this tree the following groups of processors selected and the paths for data are shown below.

	1 - 16 - 24 - 17 - 2
0 - 16 - 1	2 - 17 - 3
3 - 17 - 24 - 28 - 25 - 18 - 4	5 - 18 - 25 - 19 - 6
4 - 18 - 5	6 - 19 - 7
7 - 19 - 25 - 28 - 30 - 29 - 26 - 20 - 8	9 - 20 - 26 - 21 - 10
8 - 20 - 9	10 - 21 - 11
11 - 21 - 26 - 29 - 27 - 22 - 12	13 - 22 - 27 - 23 - 14
12 - 22 - 13	14 - 23 - 15

15 - 23 - 27 - 29 - 30 - 28 - 24 - 16	17 - 24 - 28 - 25 - 18
16 - 24 - 17	18 - 25 - 19
19 - 25 - 28 - 30 - 29 - 26 - 20	21 - 26 - 29 - 27 - 22
20 - 26 - 21	22 - 27 - 23
23 - 27 - 29 - 30 - 28 - 24	25 - 28 - 30 - 29 - 26
24 - 28 - 25	26 - 29 - 27
27 - 29 - 30 - 28	29 - 30
28 - 30 - 29	30 - 28 - 24 - 16 - 0

Notice carefully that the conflicting processor on the level 3 tree was the leaf on the left most end of the tree. In a level 4 tree, we can find 2 + 1 level 3 trees as shown in the figure below.



In a level 4 tree we have three processors which have conflicts if they send their data items out. From this, the following hypothesis can be made.

For a tree of level i there will be $(2^{i-3} + 1)$ conflicts if the data is to be transferred according to the law $x \rightarrow x + 1$. This hypothesis can be proved very easily by using induction. Thus in one message transfer stage, we

can move about $7N/16$ data items. If the data has to be moved from $x \rightarrow x + \sqrt{N}$, simulate a grid network. To do this, we will once again divide the processors into K groups on N/K processors each. Only one of these groups will be allowed to transfer data at a time. It is very easy to see that the bottleneck occurs at processors at level i and $i-1$. Performance of the moves, as I did for the ring network above, will show that at most \sqrt{N} data items arrive at these nodes simultaneously. Because only one of them can go through at a time, we will need \sqrt{N} complete transfer stages to simulate this move or in one move $(N/K)/\sqrt{N}$ or \sqrt{N}/K data items can be transferred to their destination without conflicts.

During a shuffle exchange move, the tree network has to move data from processor x to processor $2x \bmod N$. To perform this move on the tree, we will once again divide the processors into groups of N/K each with only one group transferring data at a time. It is obvious that all the processors whose number is greater than $N/2$ will be transferring data to one of their sons. (We are assuming two sons per father). Therefore, there are no conflicts for them. Among the processors numbered 0 to $(N/2 - 1)$ there are conflicts present. Simulating the moves, as we did, for the $x \rightarrow x + 1$ move, we find that there are two conflicts for a tree of level 3. With an increase in every level of the tree, the number of conflicts double. Thus,

for a tree of level 4, we will have 4 or 2^{i-2} conflicts for every N/K data items. Since we are using $K = 2$, this leads to $N/2$ minus 2^{i-2} . Because $N = 2^{i+1}$, then $N/2 - N/8 = 3N/8$.

BIBLIOGRAPHY

1. A.V. Aho and J.D. Ullman, "Dynamic Memories with Rapid Random and Sequential Access," IEEE Trans. Comput., Vol. C-23, pp. 272-277, March 1974.
2. G.H. Barnes et al., "The ILLIAC IV Computer," IEEE Trans. Comput., Vol. C-17, pp. 746-757, August, 1968.
3. M.J. Flynn, "Very High Speed Computing Systems," Proc. IEEE, Vol. 54, pp. 1901-1909, December, 1966.
4. H.S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. Comput., Vol. C-20, pp. 153-161, February, 1971.
5. W.M. Gentelman, "Some Complexity Results for Matrix Computations on Parallel Processors," Journal of the Association for Computing Machinery, Vol. 25, No. 1, pp. 112-115, January 1978.
6. J.A. Brzozowski and E.J. McCluskey, Jr., "Signal Flow Graph Techniques for Sequential Circuit State Diagrams," IEEE Trans. Electronic Computers, Vol. EC-13, pp. 67-76, April, 1963.
7. T. Agerwala and B. Lint, "Communication in Parallel Algorithms for Bodean Matrix Multiplication," Proc. 1978, Int. Conf. Parallel Processing, pp. 146-153, August, 1978.
8. S. Bokhari, "Dual Processor Scheduling with Dynamic Reassignment," IEEE Trans. Software Engineering, Vol. SE-5, pp. 341-349, July 1979.
9. C. Thompson and H. Kung, "Sorting on a Mesh Connected Parallel Computer," Commun. Ass. Comput. Mach., Vol. 20, pp. 263-274, April 1977.
10. C.D. Thompson, "Generalized Connection Networks for Parallel Processor Intercommunication," IEEE Trans. Comput., Vol. C-27, pp. 1119-1125.
11. John H. Reif and Paul G. Spirakis, "Real Time

Synchronization of Interprocess Communications."
ACM Transactions on Programming Languages and
Systems, Vol. 6, No. 2, pp. 215-238, April 1984.

12. Otto C. Judich and Clinton R. Foulk, "Compilation of
 Acyclic Smooth Programs for Parallel Execution."
ACM Transactions on Programming Languages and
Systems, Vol. 3, No. 1, pp. 24-48, January, 1981.
13. Ricart G. and Agrawala A.K., "An Optimal Algorithm for
 Mutual Exclusion in Computer Networks."
Communications ACM, Vol. 24, pp. 9-17, January
 1981.
14. K. Batcher, "Sorting Networks and Their Applications,"
 in Proc. AFIPS Spring Joint Comput. Conf., pp.
 307-314, 1968.
15. L. Ford and D. Fulkerson, "Flows in Networks."
 Princeton, NJ: Princeton University Press, 1962.
16. L. Csanky, "Fast Parallel Matrix Inversion
 Algorithms," SIAM J. Comput., Vol. 5, pp. 618-613,
 Dec. 1975.
17. B. Lint and T. Agerwala, "Design and Analysis of
 parallel Algorithms." IEEE Trans. on Software
 Engineering, Vol. SE-7, No. 2, March 1981.
18. M.C. Pease, "Matrix Inversion Using Parallel
 Processing." J.ACM, Vol. 14, 1968, pp. 757-764.