6-1995

Modeling and Development of a Cellular Micro-Kernel

K. Balaji
Western Michigan University

MODELING AND DEVELOPMENT OF A CELLULAR MICRO-KERNEL

by

K. Balaji

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science in Engineering
Department of Electrical and Computer Engineering

Western Michigan University
Kalamazoo, Michigan
June 1995

ACKNOWLEDGEMENTS

I would like to acknowledge my indebtedness to the people who have contributed in various ways to this study.

First of all, I wish to express my profound gratitude and sincere thanks to my research adviser Dr. Richard Taylor, for the inspiration, instruction and invaluable guidance given to me, for without his help this thesis would not have been possible. I also like to thank Dr. Thomas Piatkowski, Dr. John Kapenga and Dr. Garrison Greenwood for their effective and constructive criticism in order to make this thesis to the perfection of highest order.

Second, I express love bounded respect to my father Sri. P. Karuppiah, mother Smt. K. Ramashwari and my brother K. Muralidaran, whose foresight, determination and vision paved the way for all the opportunities that I am getting throughout my life.

Finally, I dedicate this thesis to Lord Krishna, for he who undergoes the pleasure and pain, gain and loss, victory and defeat of this body of mine.

K. Balaji

MODELING AND DEVELOPMENT OF A CELLULAR MICRO-KERNEL

K. Balaji, M.S.E.

Western Michigan University, 1995

A novel cellular load distribution strategy was designed and implemented on transputers by Dr. Richard Taylor and Dr. Don Goodeve at the University of York, United Kingdom. Based upon the above proposed strategy, a statistical model was developed using the Cellular Automata Theory. The theoretical model drives the implementation process of the cellular micro-kernel on the nCUBE 2S system. The load distribution and the performance characteristics for snake and L type transmission modes, and for uniform and non-uniform distribution of destination processors of the micro-kernel has been measured and compared with our theoretical model. The results show good agreement between experimental and statistical model.

TABLE OF CONTENTS

Table of Contents---Continued

LIST OF FIGURES

List of Figures---Continued

List of Figures---Continued

CHAPTER I

INTRODUCTION

High speed computers have been traditionally used in compute intensive application in science and engineering such as numerical weather forecasting, oil exploration, seismic data analysis, computational fluid dynamics and computational physics to name a few. More recently these computers have been applied to non-traditional fields such as advertising, animation and industrial computer aided design. Developments in VLSI technology have enabled spectacular advances in computer performance thus allowing scientists not only to study models of novel scientific and engineering problems, but also to construct larger more accurate models of existing problems.

Conventional means of increasing the performance of high speed computers have been through the use of more advanced techniques, among others, very large scale integration, faster switching circuits and denser packing methods. While there is still scope for speed enhancement through these conventional means, future advances are unlikely to be a spectacular as in the past. Physical and technological factors bound the maximum performance achievable with a single processor (Mead & Conway, 1980). For instance, in

VLSI circuits, the speed of light limits the speed of a travelling signal. Also computation error may arise due to electrons 'tunnelling' across the insulation if data lines are placed too close in integrated circuits.

Despite these limits the demand continues to rise both for computing power for large applications and for cost efficient computing platforms. Such demands have prompted research into new architectural approaches to providing more computer power as opposed to more traditional once above. An obvious approach that has received a lot of attention is parallel processing. A system architecture in which large number of conventional processing elements, communicating with each other through some interconnection network, and cooperating with each other in a coordinated way to solve large problem fast (Almasi & Gottlieb, 1989). Given current and foreseeable development in hardware technology, massively parallel processing system are technically and economically feasible. These systems aim to provide both increased performance and better price/performance ratios for wide range of applications.

A major concern is the provision of software support for effective use of parallel processing systems. Many crucial problems in parallel systems programming are either unsolved or partially solved. More difficulties confront a would-be programmer of parallel hardware than those encounter in conventional sequential computers. The three

key areas of research in parallel programming of parallel computer systems are (Gajski & Peir; 1985, Haynes, Lau, Siewiorek & Mizell, 1982; Hwang & Briggs; 1985, Almasi & Gottleb, 1989; Klietz, Malevsky & Chin-Purcell, 1994):

1. Specification of parallelism where researchers are concerned with developing parallel languages with special constructs that allows expression and packaging of parallelism, support synchronization and communication between application modules, and provide support for distributed data structures. Ideally these languages should shield the programmer from architectural details.

2. Exploitation of parallelism which embraces a number of issues in design and used of operating systems and compilers for parallel processing systems. Briefly these includes: load decomposition, the partitioning of applications into smaller modules called processes to tasks; process creation and management; process distribution among the processors; supervision of process synchronization and inter-processor communication; and, management of high-bandwidth input/output systems.

3. Support environments and tools which assist in program de-bugging, run-time profiling and tracing of non-determinism. In addition, they may provide facilities for performance analysis perhaps through interactive graphical user interfaces.

We are concerned with the general domain of software

tools for effective exploitation of parallelism. More specifically, in this thesis we will develop and analyze a strategy that effectively distributes load (Goodeve & Taylor, 1992) using cellular automata theory in nCUBE 2S system without incurring large communication and computation overheads.

Cellular Approach to Parallel Computer Design

Cellular automata theory was first introduced by von Neumann as a model for biological systems (Codd, 1968, Burks, 1970). He envisaged this property as being of great importance in fault tolerant computing systems (Neumann, 1956). Cellular automata (CA) usually consist of an infinite array of interacting finite state machines. Such automata can be considered at two levels of complexity. At the simple, locally interacting level the automata allow mathematical analysis. At the more global level, in which the whole system is considered, they exhibit surprisingly complex phenomena. Their power and versatility are attributable to these two factors (Wolfram, 1986).

Cellular Automata; The Abstract Mathematical Model

Cellular automata are based on the notion of discrete space, discrete time and cell states idealized as a finite set of discrete values. The cellular space is a large, usually infinite, n-dimensional regular lattice of homoge-

neous sites, or cells. Each cell and a small number of adjacent cells form a local neighborhood set. The cell interacts exclusively with cells in its neighborhood set.

In addition each cell has a discrete variable whose value is called the state of the cell. In the (possibly) infinite cellular space only a finite number of cells will be active, the rest will be in a non-active state. In general, a quiescent cell surrounded by similar cells will remain in a non-active state. Thus, cellular automata theory generally considers the finite set of active cells and their boundary with the non-active state regions.

Cells' states are updated synchronously throughout the cellular space according to a global clock operating in discrete time intervals or cycles. The state transition function is a finite set of simple rules. In one cycle the state depends completely on the state of the cell and its neighborhood cells from the previous cycle. The interconnection graph of the cells are temporally static, the nodes and their interconnections do not change with time, and are spatially regular, i.e. the graph has a simple regular geometry. Extension of these fundamental principles of cellular automata theory include such features as complex non-deterministic transition function and novel neighborhood connection (Wolfram, 1986; Demongeot, Goles & Tchuente, 1985). A number of characteristics are common in simple cellular automata. Presenting a set of simplifying features,

they facilitate analysis and simulation while providing restrictions on the overall cellular automata system.

The Cellular Paradigm for Parallel Processing Systems

Complex physical, biological and chemical systems have been modelled by mapping them into cellular automata (Lindermayer, 1968). The computational properties of cellular automata have been studied extensively (Codd, 1968; Burk, 1970; Shannon & McCarthy, 1956). Cellular automata have been used as a basis for parallel processing computing. Example include highly parallel sorters, multipliers, prime number sieves, and pattern recognition in 2-dimensional arrays (Wolfram, 1986; Hillis, 1984).

Physically, the implication of the ideal mathematical definition is an infinite regular interconnection of homogeneous cells. Each node has a small memory unit to store the cell's state and a processor to compute the transfer function and manage the neighborhood information. In the cellular paradigm two classes of machines may be identified; cellular logic machines and connectionist machines. The difference between these two is that while cellular logical machines simulate the cellular space in memory, the connectionist machine implement the cellular space in a physical network of processors.

Cellular Logic Machines

Hardware simulation of physical cellular array has been, until recently, impractical to undertake, hence the historical attractiveness of cellular logic machines. These are special purpose machines for emulation of the cellular automata. They use single (or multiple) high speed processors and user-defined transition functions to evaluate sequentially the cell state stored in the memory of an *N X N* array. Hence, while not strictly in physical structure, they are cellular in functionality. Cellular Automata Machine (CAM)(Toffoli, 1984) is a most recent system. CAM's design objective was to provide economical, high performance hardware to permit intensive research in abstract cellular automata. The CAM can simulate non-uniform, non-deterministic cellular automata with transaction functions that may be both change within a single run for simulation time, or vary from one region of cellular space to another.

The Connection Machine

Hillis' connection machine (CM)(Hillis, 1985) sprang from the NETL project, an artificial intelligence system for the storing of, and performing deductions on, a knowledge base represented as a semantic network (Fahlman, Hinton & Sejnowski, 1983). In NETL, noun-like objects were represented on node cells and their inter-relationships by link

cells. Each cell has a simple processor and is connected to a small memory unit.

The connection machine is a more general purpose machine not only capable of different searching methods, but also applicable to a wide class of problems; especially those involving memory interaction computation (Waltz, 1987). The connection machine has up to 64K processing elements each with a memory unit. Although physically connected in a simple two dimensional grid, each cell could be configured by the software to be virtually connected to all the other by use of message passing scheme.

With simple processors and interconnection hardware an economically feasible connectionist machine is brought about to explore parallelism. This is a key connectionist machines' concept and their major advantage. Their computational power increases almost linearly with the problem size because, depending on the availability of processor and nature of the problem, the execution time is nearly constant with respect to the size of the problem (Hillis, 1984).

Cellular Structures of Our Model

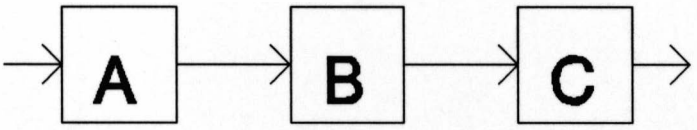
The cellular space in our model at the network consists of a static regular topology of interconnected homogeneous nodes. We use the basic cellular automata theory as outlined earlier. Each cell is connected by directed links to a set of cells called its connection set. All

interactions between the cells occur via these links.

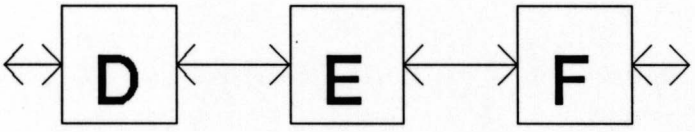
Each cell has a distinct state from a finite alphabet of such states. After each time cycle each cell communicates its state to some or all of the cells connected to it. These cells belong to its negative neighborhood set. At the same time, the cell receives the states of all or some of the cells connected to it. Similarly these cells belong to its positive neighborhood set. the positive and negative neighborhood sets are not necessarily identical; in some models they might be specified nets (Morgan, 1987).

To illustrate the concept of positive and negative neighborhood, consider two assembly as shown in Figure 1.a and Figure 1.b. The first, which station A, B, and C, has load moving from left to right. Thus, in one cycle, the state at station B will depend on whether A is functional or not. A is thus in the positive neighborhood of B since it contributes positively to the next state at B. Conversely, B's state affect the next state of C. If B were to malfunction then there would be no load to C. We say that C is in the negative neighborhood of B. If we have an assembly, like Figure 1.b, where load flows in both directions, then both neighbors of E would be in its negative and positive neighborhoods. Status information is thus passed to all neighbors; both the connectivity and neighborhood sets are equivalent.

When this cellular automata paradigm is used to model



a



b

Figure 1. Neighborhoods in a Linear Array.

load distribution by cellular interactions, the load flow time depends on the communication time. Since the relative position at the local level of the neighbors are same with respect to a particular processor, both positive and negative neighborhood sets tend to remain the same. Depending on the neighbor which ever closer to the destination processor, the transmission of the load is considered.

The processor is described using standard cellular automata mathematical notation. The model closely follows the works of cellular automata (Burk, 1970; Codd, 1968) and description of cellular structures for operating systems (Wendler, 1981). Though the description applies to n-dimensional cellular structure, the example given, and our performance model, use 2-dimensional *N1 X N2* toroidal (warped around) mesh, like the one shown in Figure 2.

The Test Bed

In this thesis the theory of load distribution on a parallel system was modeled using cellular automata theory, as expressed in the earlier part of this chapter. The obtained theoretical model is tested by implementing the model on the nCUBE 2S system which is comprised 128 processors configured as a 7D hypercube, which is used as our test bed.

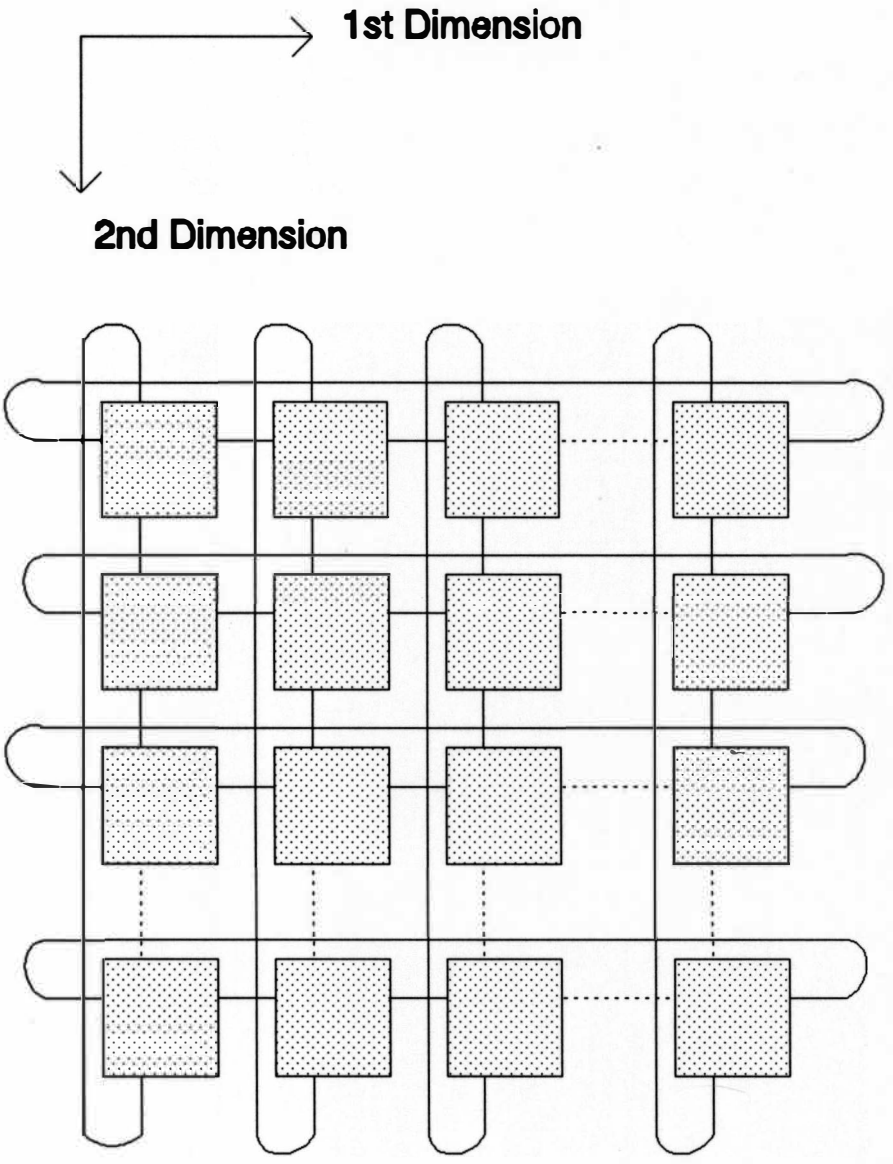


Figure 2. A N x N Toroidal Four Neighbor Mesh.

Hypercube Architecture

Massively parallel processor (MPP) is a coordinated set of hundreds or thousands of computer processors that share a fast communication network. MPP's can be classified based on their memory addressability (shared or distributed) and the manner in which they execute instructions (Single Instruction Multiple Data (SIMD) or Multiple Instruction and Multiple Data (MIMD)). Our nCUBE 2S system falls into the category of MIMD system, in which processors independently execute different instructions on different data at a given time cycle. Our nCUBE 2S system follows the hypercube architecture. Hypercube architecture have much studied (Almasi & Gottlieb, 1989); several companies, including Intel, NCUBE and FPS, have marketed machines having this topology. In a k-dimensional hypercube we have $N = 2^K$ nodes, each of degree K.

The n-cube or n-dimensional hypercube Q_n is defined recursively in terms of the cartesian product of two graphs as follow:

$$\begin{aligned} Q_1 &= K_2 \\ Q_n &= K_2 \times Q_{n-1} \end{aligned} \qquad (1.1)$$

Thus the n-cube, Q_n may also be defined as a graph whose node set V_n consists of the 2^n n-dimensional boolean vectors, i.e., vectors with binary coordinates (Harary, Hayes & Wu,

1988). Figure 3 shows the n-cubes for $n \leq 3$ with appropriate boolean vectors as node labels. From the Figure 3, we can observe that each neighboring node differs by one and only by one bit with each other. A graph G = (V, E) has p = |V| nodes and q = |E| edges, and is said to have order p and size q. Thus, the order of Q_n is 2^n and its size is $n\ 2^{n-1}$.

The n-dimensional hypercube has 2^n nodes. To each node we associate a n-bit binary label; adjacent nodes have binary labels that differ in one and only one bit position.

Let $s = s_{n-1}...s_1s_0$ and $d = d_{n-1}...d_1d_0$ be the binary label of the source node and destination node, respectively. Further, let $v = v_{n-1}...v_1v_0$ be the label of any node along the route. Then the routing algorithm (Greenwood, 1995) is

1. compute $r = s \oplus d$
2. let $v = s$ and $i = 1$
3. if $r_i = 1$, route from v to $v \oplus 2_{i-1}$
4. $i \leftarrow i + 1$
5. if $i \leq n$ go to step 3. Else, EXIT.

An example for this algorithm at work is shown in Figure 4 with s = 0110, d = 1101, and n = 4.

1. $r = s \oplus d = r_4r_3r_2r_1 = 1011$
2. $v = s$ and $i = 1$
3. $r_1 = 1 \Rightarrow$ route from s to $v = s \oplus 2^0 = 0111$
4. $r_2 = 1 \Rightarrow$ route from v to $v \oplus 2^1 = 0101$
5. $r_3 = 0$ so skip the next dimension

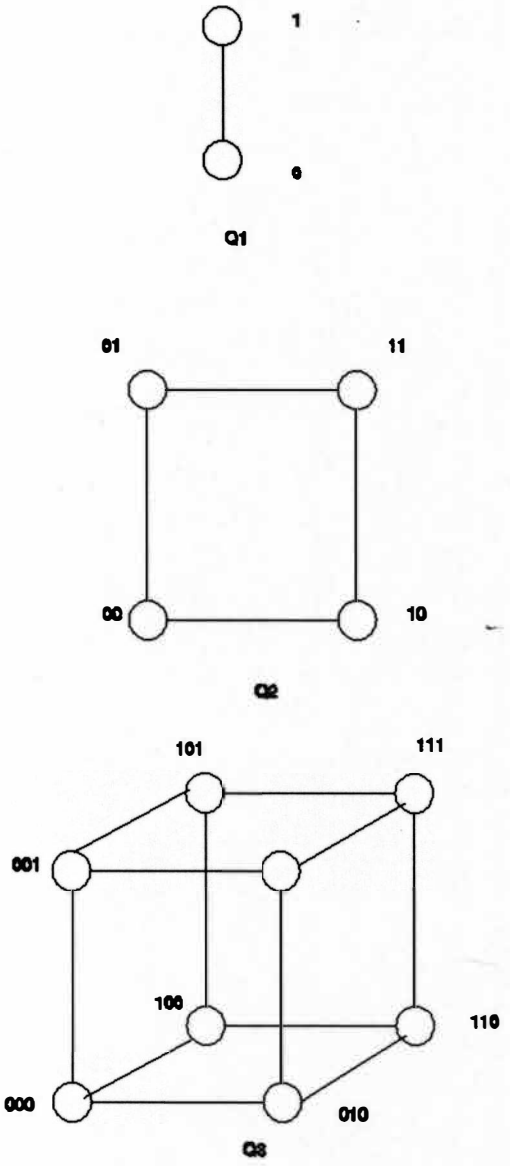


Figure 3. n-cube Graphs for n = 1, 2, 3.

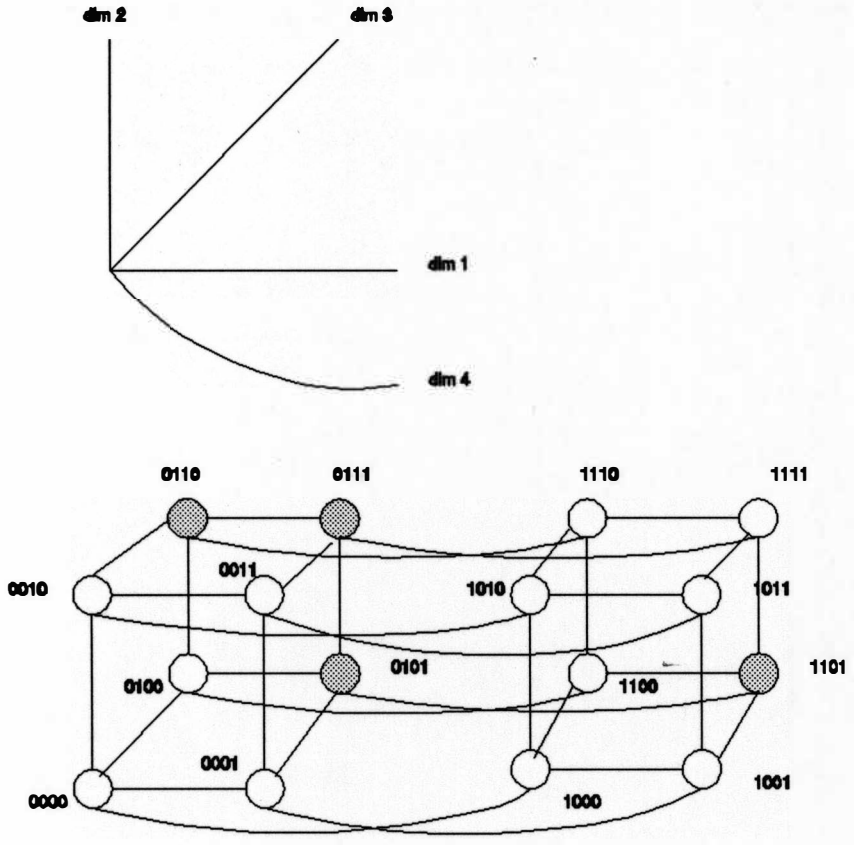


Figure 4. Routing Algorithm at Work.

6. $r_4 = 1 \Rightarrow$ route from v to $v \oplus 2^3 = 1101 = d$.

nCUBE 2S System Architecture

The processing nodes of our nCUBE 2S system are based on a custom single-chip processor, which combines 64-bit CPU, 64-bit floating point unit, and a message-routing unit onto a single 500,000-transistor chip. The nCUBE 2S processors' architecture is shown in Figure 5. The processor architecture is VAX-like. With 20MHz clock, the processing node's performance is 7.5 MIPS and 2.4 MFLOPS (3.5 MFLOPS single precision). Node memory is between 4 MBytes to 16 MBytes. The maximum configuration is 8192 nodes (a 13D hypercube). The main philosophy of nCUBE 2S system is to develop a processor, designed specifically for parallel processing, that balances computation with communication. The nCUBE 2S processor realizes this philosophy by integrating communication channels with its processing facilities. Each processor includes 14 bidirectional communication channels: 13 for the interprocessor network - supporting a hypercube with up to 8192 processor - and one for I/O. When multiple nCUBE 2S processors are configured in a hypercube network, the processor architecture provides unmatched communication bandwidth (Voigt, 1994). And because each processor includes its own communication facilities, adding processors to a system increases computation speed,

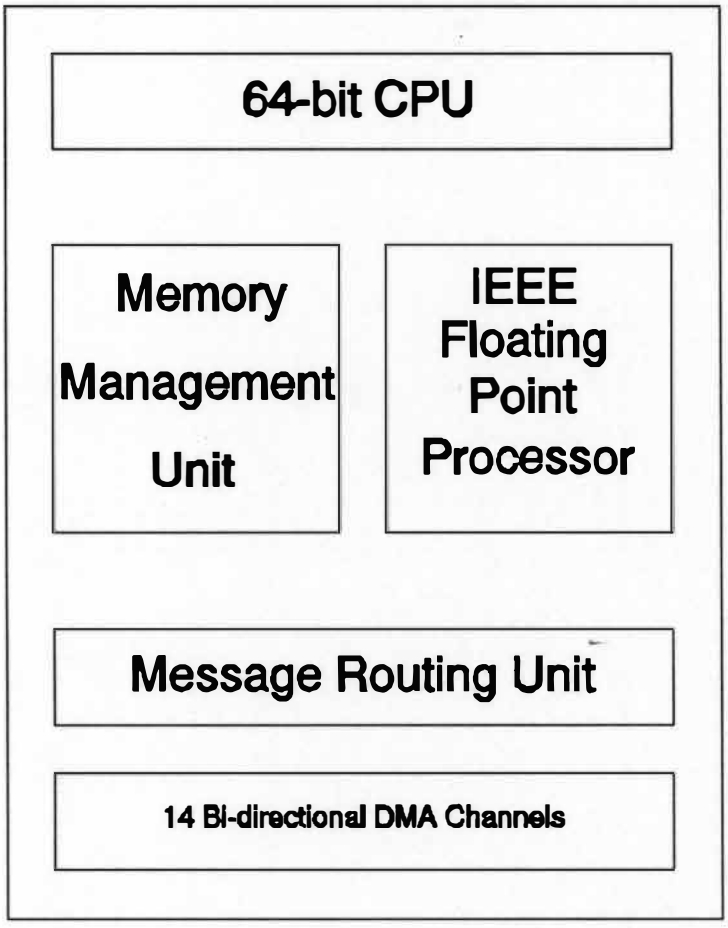


Figure 5. The Architecture of nCUBE 2S Processor.

communication bandwidth, and I/O bandwidth. An nCUBE 2S system supports C and fortran programming. The host computer runs AXIS, a UNIX-based operating system that manages to make the whole machine look like one distributed file system. VERTEX is a small (< 4KB) kernel in each node; its primary function is internode communication (message routing and store-and-forward buffering).

Various authors (Voigt, 1994; Gubbala & Singh,1993; Schmidt, Dick, Forbes, & Tasker, 1992; DeBenedictis & delRosario, 1992; Palmer, 1988) have studied the communication aspects of nCUBE 2S system. In our thesis we study the communication aspect using cellular automata theory. During the implementation processes of the cellular theoretical model on the nCUBE 2S system, we squash the cubical architecture of the nCUBE 2S system (of the used hypercube) to a toroidal (wrapped-around) 2-dimensional mesh architecture to entertain our experimentation.

Thesis Outline

In Chapter II we formulate a statistical model based upon the system architecture that we are going to use to perform the experiment with the theory of cellular automata. The formulated model is numerically analyzed and studied. Conclusions were drawn about from the numerical results, so as these results can be compared with the experimental results. In Chapter III we describe the method, design, and

technique by which the experiment is performed in the nCUBE 2S system through the drive of the theoretical model, while Chapter IV provides a complete description of the features of the micro-kernel. In Chapter V we provide the conclusions and future work required for further development to the aspects of the micro-kernel.

CHAPTER II

STATISTICAL MODEL AND ANALYSIS

A key issue in performance analysis of parallel systems is the behavior of the interconnection network with different types and levels of messages traffic. The speed of the individual components, the type of medium and strategies used in building the protocol will be obviously have an impact on the network's communication bandwidth. However, aside from these implementation characteristics, the performance of the network depends on the architectural characteristics; such as the number of neighbors, the interconnection topology used, the size of the network, and the load distribution scheme's characteristics; such as the amount load shared in each cycle, the frequency and the amount of load exchange.

Therefore an analytical tool is needed to predict the performance of a target network for the range of communication needs of an intended application. In this chapter we develop a statistical model for the performance of two dimensional toroidal mesh networks. Though applicable to larger neighborhoods, the analysis has been applied to a four neighbor mesh. The performance of the network is evaluated in terms of relative time per byte versus the

message generation rate and length of the message (Taylor, 1995).

Cellular Load Distribution (CLD) (Goodeve & Taylor, 1992; Macharia, 1990) schemes are a class of dynamic heuristic load distribution strategies that share load in a neighborhood of cooperating processors. This analysis allows us to predict the expected relative time per byte for different levels of communications traffic due to process interaction and load transformation.

Introduction

Time delay in the nCUBE 2S system has been analyzed extensively by Voigt (Voigt, 1994) and others. In this chapter we develop a statistical model of four neighbor two dimensional toroidal mesh based on cellular automata theory. The goal is to find relationship for relative time between messages between any two processors with respect to message generation rate and message length. In our nCUBE 2S system while designing the two dimensional toroidal mesh, we consider each intermediate processor as switches having buffers. With this background, in a closed toroidal mesh no messages can be lost, hence the performance metric is the message latency or communication time delay, from which relative time per byte can be calculated. In this case the bandwidth is the product of the number of messages generated of message length and the message latency per unit time.

The Model of Operation

The mesh is modelled as a regular 2-dimensional planar 'cellular' network consisting of an *N1 X N2* array of identical nodes, or processors, or cells. Each of these nodes is connected to a fixed number of neighbor *d* by dedicated point to point bi-directional links and the edges of the array are 'warped-around'. Though applicable to other type of homogeneous 2-dimensional network the analysis here is specific to four neighbor cellular network. There are four important properties to this model, which are described as follow:

The Node

Each processor or node has unique identity and a very specific spot in the 2-dimensional network. In this model the node is modelled as a processor/memory device generating (and accepting) messages at a random rate for (from) arbitrary destination. The nodes are asynchronous in that the rate of message generation is non-deterministic and independent of other nodes. For network implementation independence, a clock is specified that allows each node to operate asynchronously in equal discrete time units or cycle. One cycle is further defined as the time taken by one message to traverse one ideal link.

The Communication Model

A store and forward scheme is used that works in this way. When a node receives a message on one of the *d* links, it compares the message's destination and the current node's coordinates; if a match is made the message is saved in the processor memory, thus a message has reached the destination. Otherwise the node saves the message and tries to pass the message on to the neighbor closes to the destination. After one such routing operation the message is one hop closer to the destination. More than one intermediate destination my be valid. Once such link is chosen at random more than one message to one link in a cycle. If the links do not have buffers all but one message will be lost. But in our case it is a network with buffer on it.

The Span

A message traverses one communication link in one cycle or hop. Hence the hopcount *h* of a particular message is the minimum number of hops between its current cells and its destination cell. The diameter *D* of a network is the maximum hopcount possible between any two cells. For example it is clear that for a *N1 X N2* toroidal mesh with $d = 4$ the diameter is equal to the dimension of the hypercube used in construction of the network (Greenwood, 1995). The well documented weakness of meshes is the high message latency,

especially with message broadcasting and hot-spot addressing (Yalamanchili & Aggarwal, 1987). However, in medium of fine grain parallel processing a large body of applications exhibit localized referencing (Ghosal et al., 1989; Wallkqvist et al., 1987; Vitanyi, 1984; Sargeant, 1987; Bunt & Murphy, 1984). To improve performance we exploit this referential locality to limit the maximum message hopcount. We constrain the possible destination nodes of a message to the span of its source cell. Thus the maximum hopcount h_{max}. In this model the toroidal mesh is of dimension $8 \; X \; 4$ (i.e $5D$ hypercube), therefore the value for $D = 5$, and h_{max} is 6.

Message Generation

All the cells in the network are identical with identical maximum hopcount h_{max}. In this model each processor generates messages of hopcount i ($1 \leq i \leq hmax$) with random hopcount probability distribution y_i, thus the traffic received from the neighbors is stochastically identical.

Message Acceptance

The view held here that the processor are connected to $d = 4$ links, and our processor has buffers to hold the messages, each node can accept multiple messages. Thereby we follow multiple-accepting model (MAM) in one cycle.

Performance Metrics and Parameters

The parameters associated with the communication in 2-dimensional toroidal mesh are as follow:

L = Length of the message,

m_g = Message generation rate per cycle = message acceptance rate = m_a,

m = The rate at which the message arrive from a particu lar neighbor,

P_t = The termination probability i.e the probability that a message received from a neighboring node is to the destination,

P_a = The probability that a non-terminal message will be accepted for re-transmission,

h_{max} = The maximum hopcount for a given warped 2D mesh.

d = Number of links,

y_i = Probability that a generated message has a hopcount i. Where y_i as used here initially has a uniform random distribution over $1 \leq i \leq h_{max}$.

By considering that no messages are lost in the *2D* grid of four neighbors, the message generation rate m_g (message acceptance rate m_a) can be expressed in terms of number of links, the rate at which the message arrive (send) from the neighbor, and the termination probability.

Hence,

$$m_g = m_a = dmP_t \qquad (2.1)$$

Hence from the above equation the rate at which the message arrive from a particular neighbor can be expressed as

$$m = \frac{m_g}{dP_t} \qquad (2.2)$$

A message may not have a hopcount greater that h_{max}. Hence at any time, say cycle i, messages from an arbitrary node $i-1$ hops away $(0 \le i \le h_{max})$ may have hopcount between 0 and $h_{max}-i$. Consequently after h_{max} cycles the hopcount distribution reaches a steady state. This distribution is given by:-

$$P_r = [hopcount = i-1] = \frac{\sum_{j=1}^{h_{max}} j dP_a^{j-i+1}}{\sum_{k=1}^{h_{max}} \sum_{j=k}^{h_{max}} j dP_a^{j-k+1}}$$

A message terminates when hopcount = 0. Moreover for our network the value for $P_a = 1$. Hence P_t is given by:

$$P_t = \frac{\sum_{j=1}^{h_{max}} j}{\sum_{k=1}^{h_{max}} \sum_{j=k}^{h_{max}} j} \qquad (2.3)$$

Substituting the above equation in the expression for m with the values for h_{max} we get

$$m = mg \frac{(15h_{max}+1)}{21d} \qquad (2.4)$$

To evaluate the time delay, the average length of the buffer must be found. Using the classical solution from the queuing theory (Kleinrock, 1975; Gray & Odell, 1970; Edwards, 1971; Abramowitz & Stegun, 1970) and the average service time of one cycle, the time spent by the message on the queues is then computed.

Consider a system consisting of just one output line on the processor. The probability of getting a message on this link is $m[(1 - P_t)/(d - 1)]$. It is obvious that the message will not return on the link it arrived on. This system is said to be in state i with probability b_i when there are i messages on the queue. Messages come from the processor (i.e the generated/transaction messages) and from the neighboring processors. Considering the former case the probability $q(i)$, of getting i messages at a particular output line is:

$$q(i) = \binom{d-1}{i} \left(1 - m\frac{1-P_t}{d-1}\right)^{d-1-i} \left(m\frac{1-P_t}{d-1}\right)^{i} \qquad (2.5)$$

for $0 \leq i < d$. The first term is due to the number of possible ways i messages can be arrive from $d-1$ links. The second term is the probability of not getting any messages for this link from $d-1-i$ links. The last term is the probability of getting i messages on the link.

Considering the generated messages as well we have four distinct cases. m_g/d is the probability of getting a source processor to generate message for one link. If $i = 0$, no message is generated for this link and it is in state $q(0)$. For $0 < i < d$ the system will either have been in state $q(i)$ and get no additional message from the processor, or have been in the state $q(i - 1)$ and have received one message from the processor. If $i = d$, the system must have been in state $q(d -1)$ before receiving one message, since at most one message is sent out every cycle. For that same reason i cannot be greater than d. The probability, a_i, of getting i messages at an input line is thus given by:

$$\begin{array}{ll} (1 - m_g/d)q(0) & \text{if} \quad i = 0 \\ (1 - m_g/d)q(i) + m_g/d\ q(i-1) & \text{if} \quad 0 < i < d \\ m_g/d\ q(d - 1) & \text{if} \quad i = d \\ 0 & \text{if} \quad i > d \end{array} \qquad (2.6)$$

Given these arrival rates at the output, the state

distribution for the queue will be given by:

$$b_i = \sum_{j=0}^{j=i} a_j b_{i-j} + a_i b_0 \qquad (2.7)$$

with this state distribution the mean number of messages in the queue may be calculated and found to be:

$$\bar{b} = m + \frac{m^2 (d(1-P_t^2) - 2(1-P_t))}{2(d-1)(1-m)} \qquad (2.8)$$

Using Little's identity theorem (Little, 1961) the mean time spent by the message at an intermediate queue is the product of the mean queue length and the average service time, i.e.

$$\bar{t} = \frac{\bar{b}}{m} \qquad (2.9)$$

Now, the average message goes through $1/P_t$ processors. Thus the delay in reaching the destination takes the form

$$T = \frac{\bar{b}}{mP_t} \qquad (2.10)$$

This expresses the delay for the one way trip in the 2-dimensional toroidal mesh for communication between any two processors.

Numerical Analysis and Conclusions

Numerical calculations were carried for the above formulated statistical model. From the calculations, the relative time between messages between any two processors in the *8 X 4* 2-dimensional toroidal mesh were calculated from the time delay and message generation equations for message generation rates of 20000, 40000, 60000, 80000, and 100000 bytes and for message lengths of 8, 16, 32, 64, 128 bytes. The results obtained were graphically interpreted. Graphs were plotted for relative time per byte against message generation rate, and message length. In the graph plotted for relative time per byte against message generation rate, shown in Figure 6, we can observe that as the message generation rate increases the relative time per byte increases. This gives us a clear picture that more the message per cycle (traffic) the more the time taken to reach the destination, hence more the relative time per byte. Also from the statistical model we can find from Б that after certain value of message generation rate or message length saturation is expected due to the limitation of buffer size. In the graph plotted for relative time per byte against message length, from Figure 7, we can observe that as the message length increases the relative time per byte decreases. The decreases in relative timings are due to the fact that the message length *L* are broken in to small

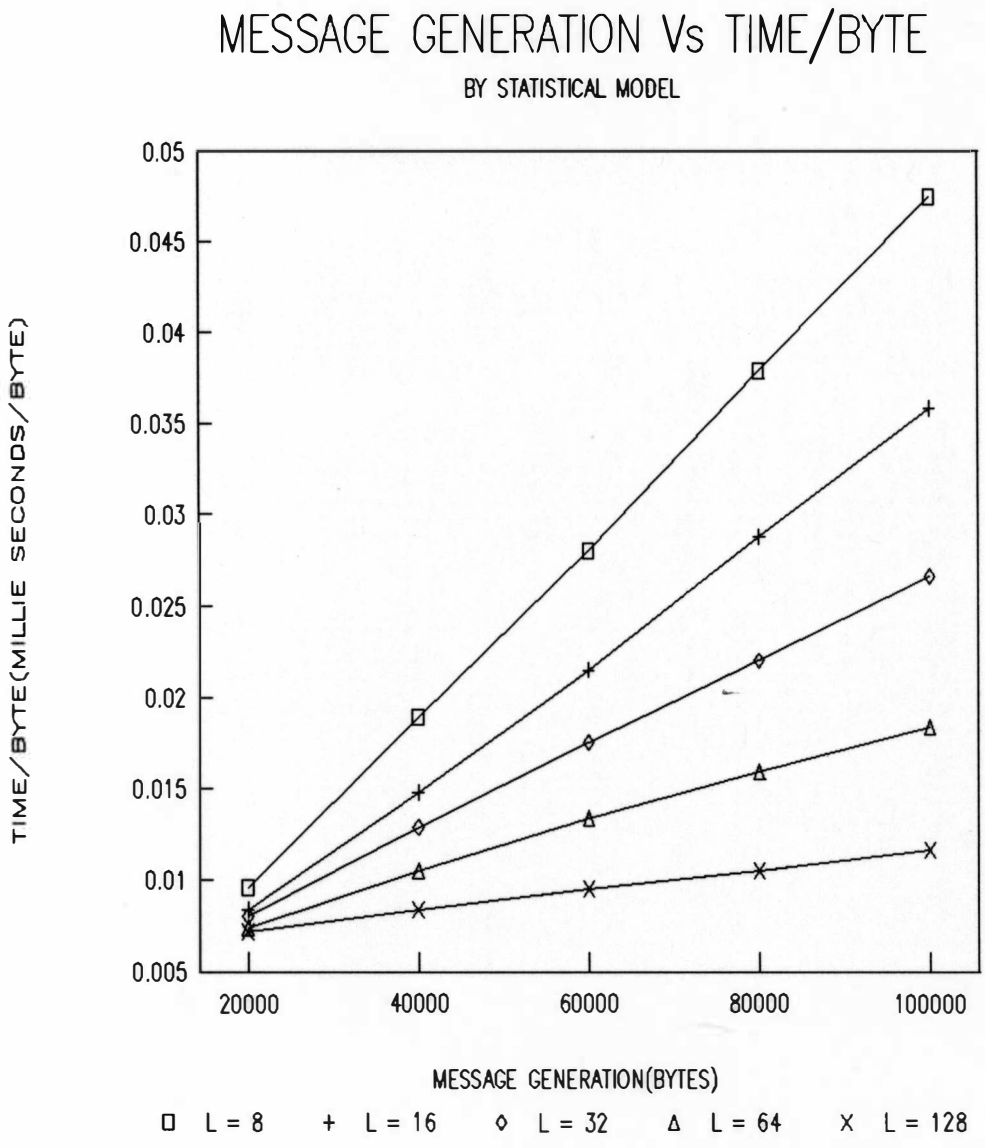


Figure 6. Relative Time per Byte Versus Message Generation Rate for the Statistical Model.

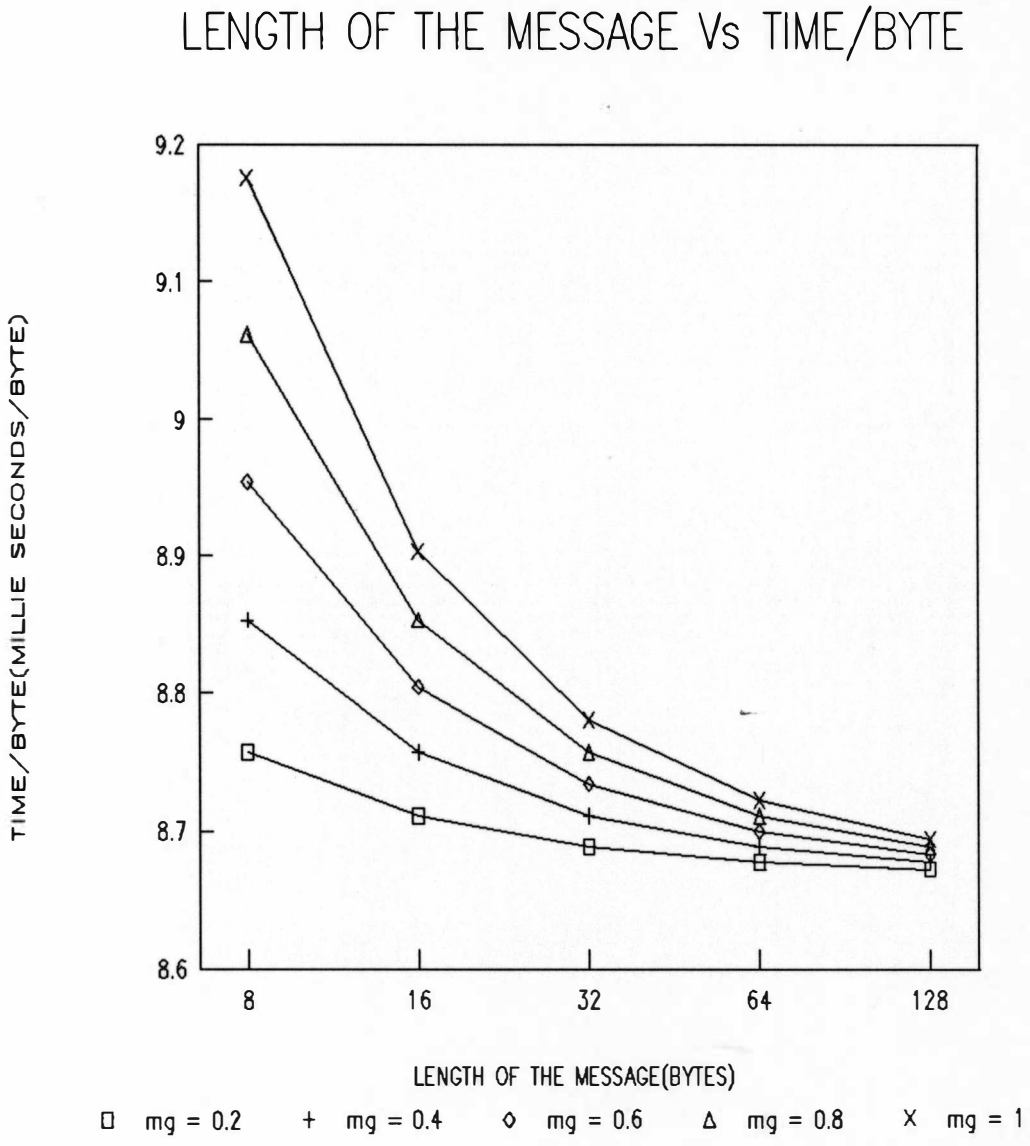


Figure 7. Relative Time per Byte Versus Message Length for the Statistical Model.

packets and the setup time for communication for these packets overlap with that of the transmission time of the previously transmitted packets, due to this, time is saved in the initialization process of communication, thereby decreasing the relative time per byte between any two processors in the given 2-dimensional mesh.

CHAPTER III

EXPERIMENTAL VERIFICATION

Intuitively, in any MPP system the faster the loads are distributed between the processors the faster the work gets done. Load distribution schemes thus, aim to move load as fast as possible from the source processor to the destination processor through intermediate transmission processors. Cellular Load Distribution (CLD) schemes allows load to pass through intermediate processors, one processor per cycle through the network.

Earlier, in the statistical model we described the modeled system as a 2-dimensional toroidal network with four neighbors. In accordance with the cellular automata theory each processor can transmit (or receive) messages to (or from) any processor in the given network, but the transmission (or receiving) of the message can take place only through any one of the four neighboring processors, which is chosen dynamically depending on which processor is closer to the destination processor.

On this basis of cellular automata's allowed mode of transmission of messages through the network, there are two distinct kinds of transportation. One is the "snake" type of transmission and the other is the "L" type of transmis-

sion. A snake type of transmission is one in which the source transmitted messages arrive to the destination through the intermediate processors in a way that each time the hop of the message take place dynamically choosing the neighboring processor such that each hop alternatively choose the x and y axis directional processors, so as the path of transmission from the source to the destination look like a track of a moving snake. On the other hand, the L type transmission, tends to transmit messages linearly on any one of the direction (either x or y directional processor) until the a match of that coordinate is made, then the messages are transmitted on the other direction until the message reaches the destination processor. Thus the transmission path looks more or like alphabet "L". But fortunately, both of these transmission modes take the same number of hopcount for transmission of messages for any specific source to the destination in a given 2-dimensional toroidal mesh. And this aspect provides a better avenue of applying cellular automata theory for our analysis, as well as staying with in the assumption that we proposed for our statistical model.

As our experiment requires a 2-dimensional toroidal mesh with four neighbors, the hypercubical architecture of our test bed nCUBE 2S system is first embedded into a 2-dimensional toroidal grid and then experiment were conducted. In this experimentation we used a toroidal mesh of *8 X*

4 (i.e hypercube of 5-dimension) of our nCUBE 2S system. The experiment was performed by transmitting messages of message lengths of 8, 16, 32, 64, and 128 bytes for both uniform distribution of destination processors, and non-uniform distribution of destination processor of $\log_2()$, $\log_3()$, $\log_4()$, $\log_5()$, and $\log_{10}()$. The delay was noted for each transmission, from which the relative time per byte were calculated. In this experimentation we want to find the round trip timing (i.e the messages are transmitted from the source to the destination processor by any (both) modes of transmission. Once the message reaches the destination processor, the source and the destination processors are interchanged so as the messages is re-routed back to the original source processor, from where the message initially initiator for transmission). This process appears more or like a "fork" process in the UNIX system, and such a process has tremendous advantage on the system programming aspect (Schaffer, 1995). On the aspect of MPP, in future this kernel can be used as a shuttle like vehicle which can carry the code modules of functional language to get distributed in the processors of the toroidal mesh to get evaluated, and then get the evaluated functions to the original processor, by doing so the efficiency, fault tolerance, and speed of the system can be improved, this is further discussed in Chapter V. The whole experimentation were carried for both

the mode of transmission, namely worm type and L type. And the experimentation was carried out for 100000 processes.

As a part of the procedure of the experiment, initially the hypercubical architecture of the test bed nCUBE 2S system is embedded into a 2-dimensional toroidal (warped around) mesh. Then the message length is set so as the required buffer size is set to transmit messages from and to any processors. Along with this, the number of the processes is also set. In this experiment the number of processes is 100000. Once all the initialization are setup in the memory of the system including the mode of transmission, distribution of the destination, the kernel is set to run and the time delay per byte are measured and stored in a file, which is subjected to graphical analysis.

In the section of the graphical analysis, we plot two kinds of graph. One relative time per byte versus message generation rate, and other with relative time per byte against message length. Figure 8 shows the plot of relative time per byte versus message generation rate for snake type mode of transmission, and uniform distribution of destination processors. From the graph we can observe that as the message generation rate increases the relative time per byte increases. This gives us typical information that more the messages are generated the more the time it is going to take for the messages to reach the destination processors. The same experiment was performed for non-uniform distribution

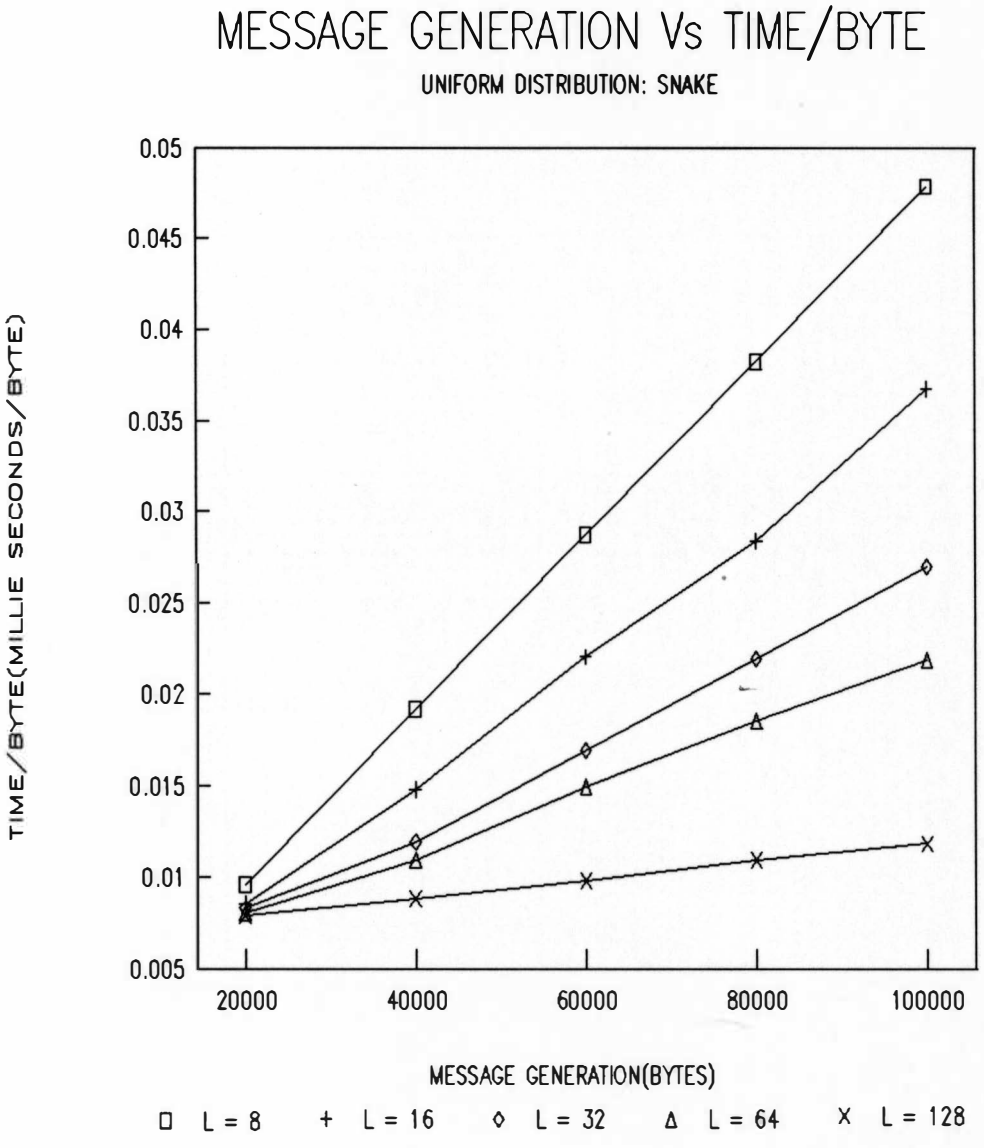


Figure 8. Relative Time per Byte Versus Message Generation Rate for Snake type Transmission, for Uniform Distribution of Destination Processors.

of destination processors. Figure 9 to Figure 13 are the plots for relative time per byte versus message generation rate for $\log_2()$, $\log_3()$, $\log_4()$, $\log_5()$, and $\log_{10}()$ based non-uniform distribution of destination nodes. From these graphs one can observe the same kind of representation as seen in the uniform distribution of processor.

The above experimentation was also carried for the L type mode of transmission of messages. Figure 14 represents the plot for the relative time per byte versus message generation rate for uniform distribution of processors. By observing the plot one can see the same sort of representation as snake type i.e., increase in relative time per byte as the message generation rate tends to increase. Figure 15 to Figure 19 represent the non-uniform distribution of the destination processors. All these graphs shows and speaks the same results as the before graphs for both snake type and L type transmission modes. Hence, it is irresistible to say that irrespective of the modes of transmission and distribution of processors the relative time per byte shall increase as the message generation rate increase.

In the second kind of graph, we plot the relative time per byte versus message length. Figure 20 is the graph plotted for snake type transmission for uniform distribution of processors. From the graph we can observe that as the message length increases the relative time per byte

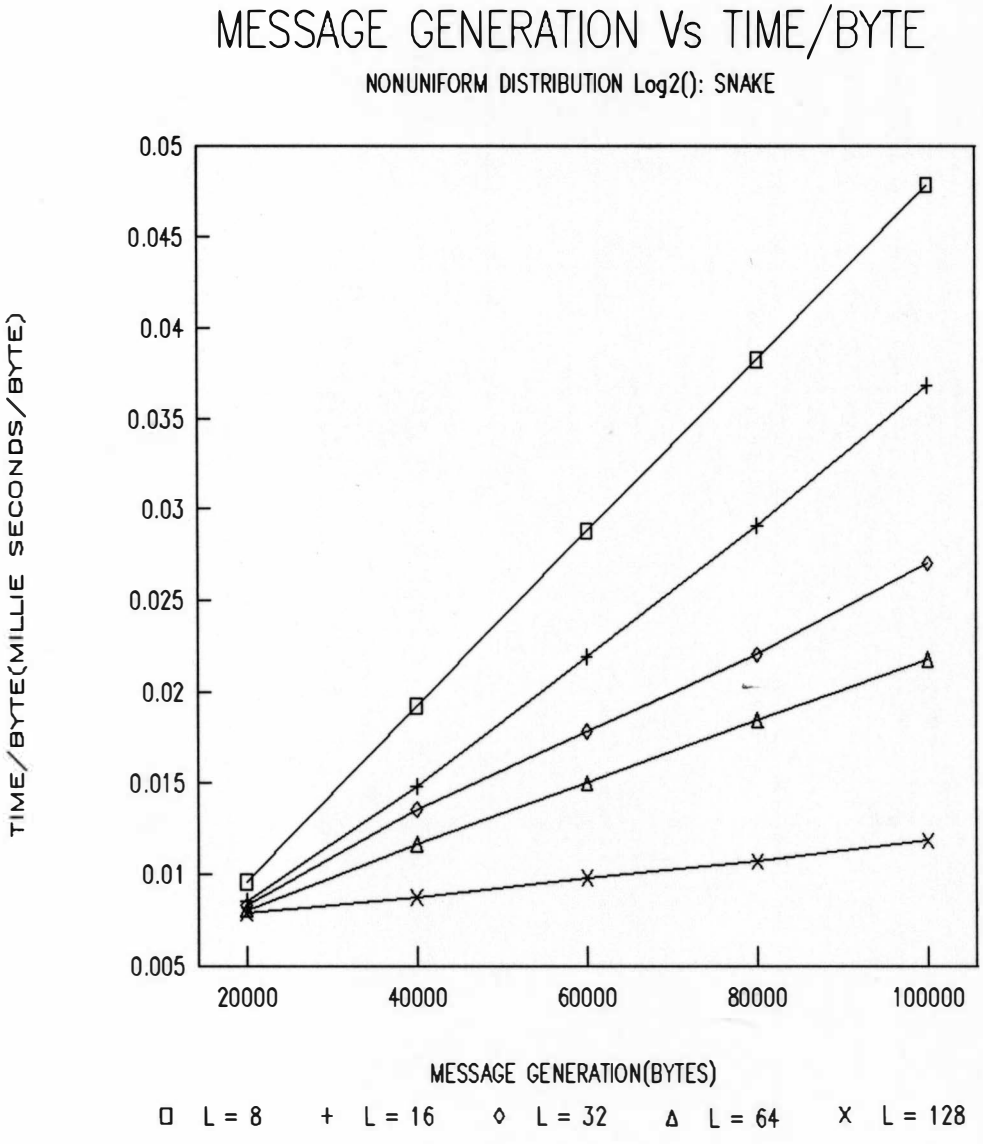


Figure 9. Relative Time per Byte Versus Message Generation Rate for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_2()$ Distribution.

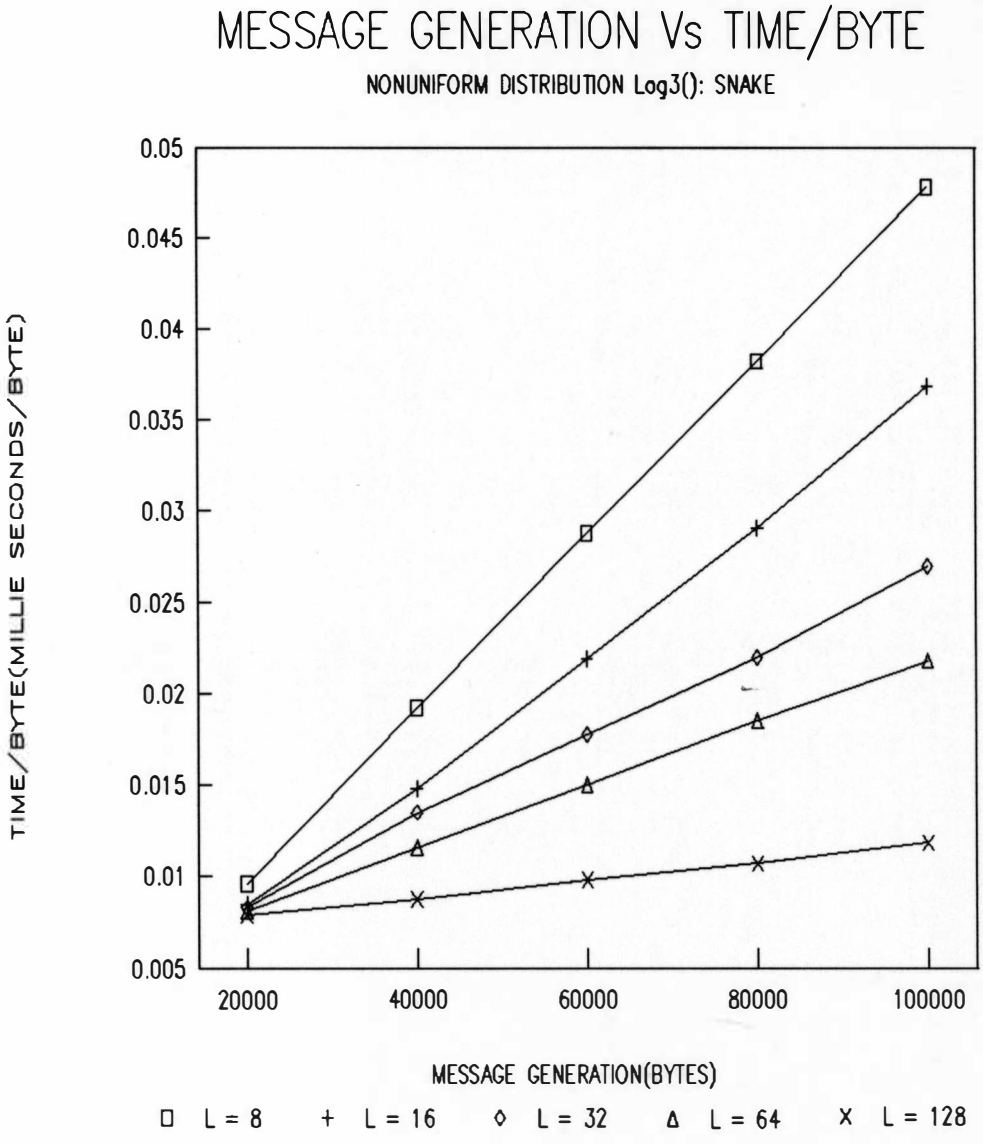


Figure 10. Relative Time per Byte Versus Message Generation Rate for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_3()$ Distribution.

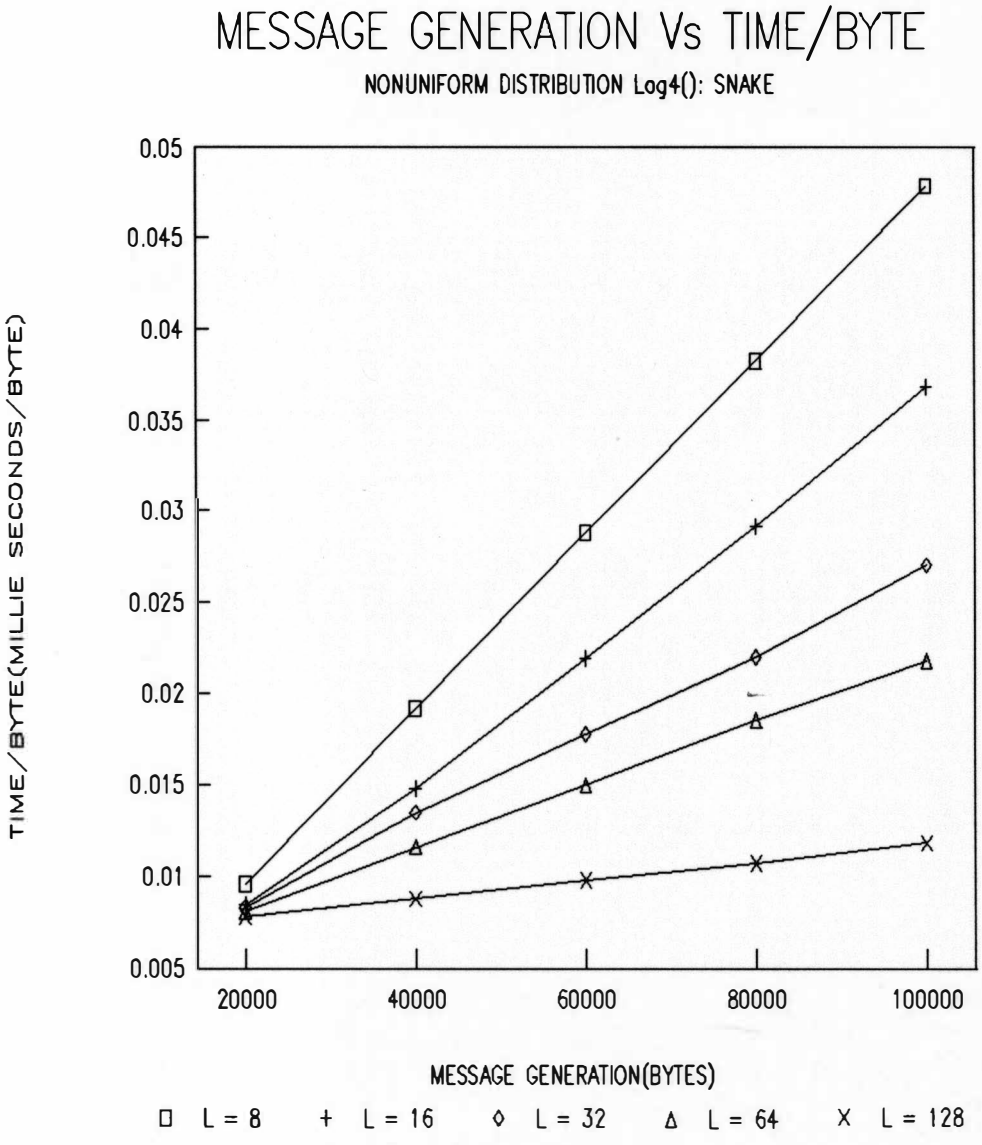


Figure 11. Relative Time per Byte Versus Message Generation Rate for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_4()$ Distribution.

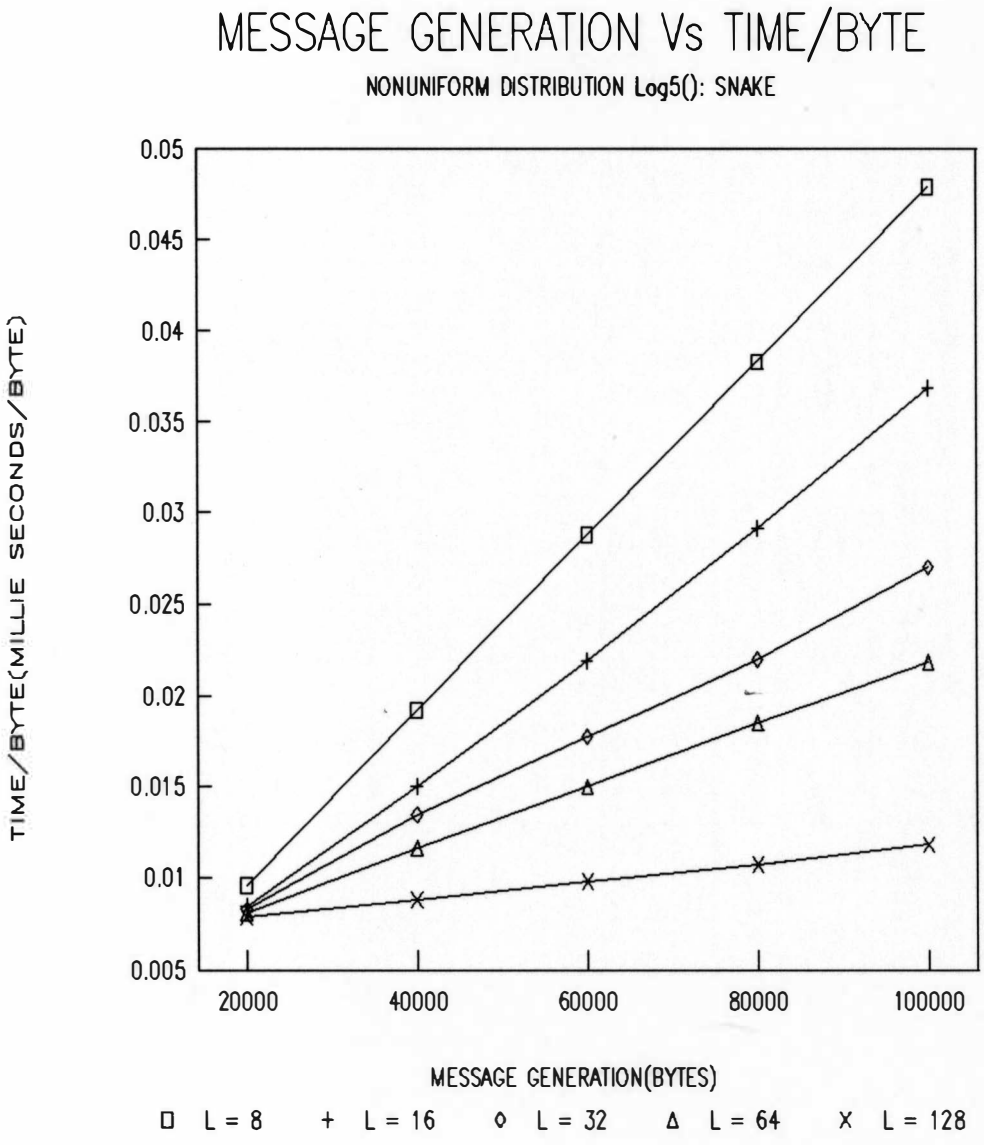


Figure 12. Relative Time per Byte Versus Message Generation Rate for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_5()$ Distribution.

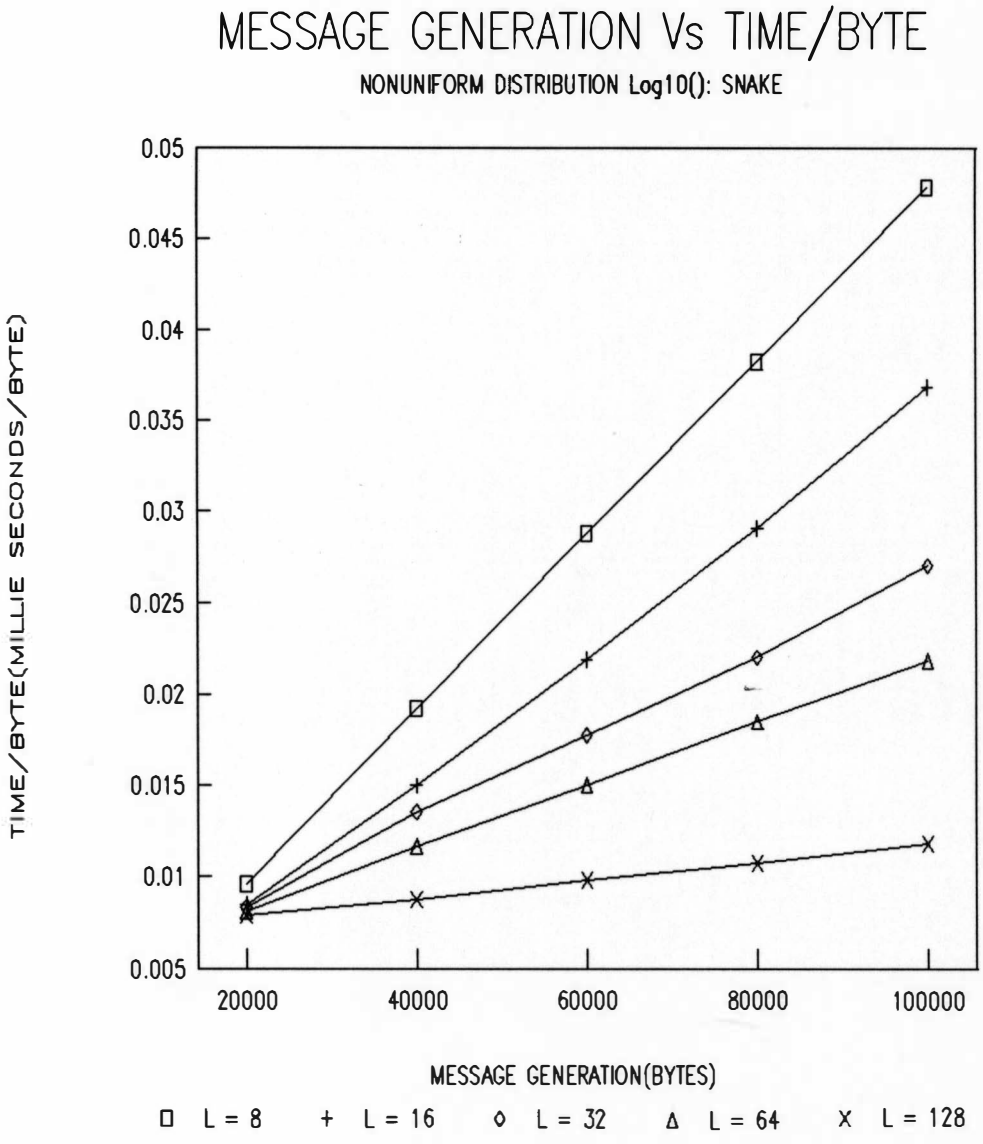


Figure 13. Relative Time per Byte Versus Message Generation Rate for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_{10}()$ Distribution.

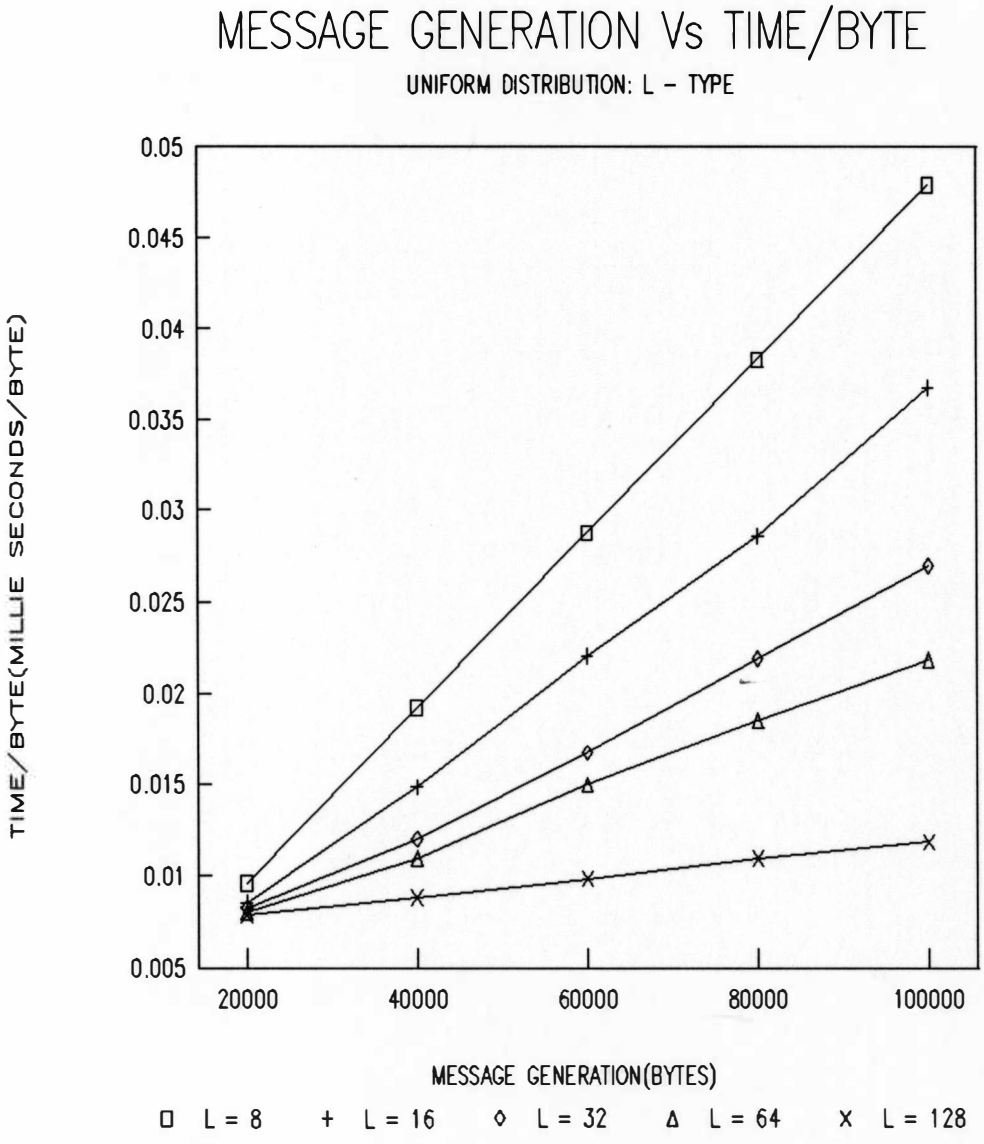


Figure 14. Relative Time per Byte Versus Message Generation Rate for L Type Transmission, for Uniform Distribution of Destination Processors.

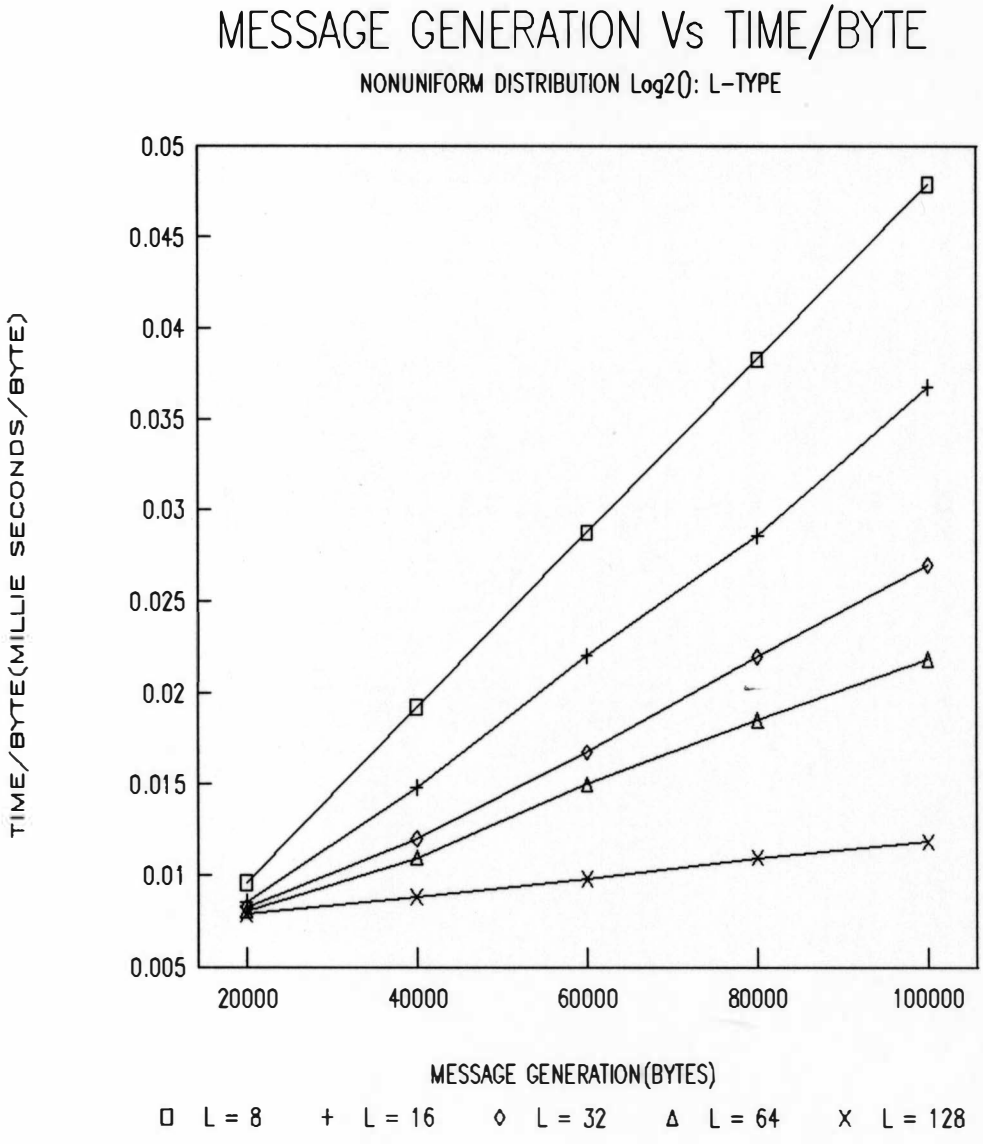


Figure 15. Relative Time per Byte Versus Message Generation Rate for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_2()$ Distribution.

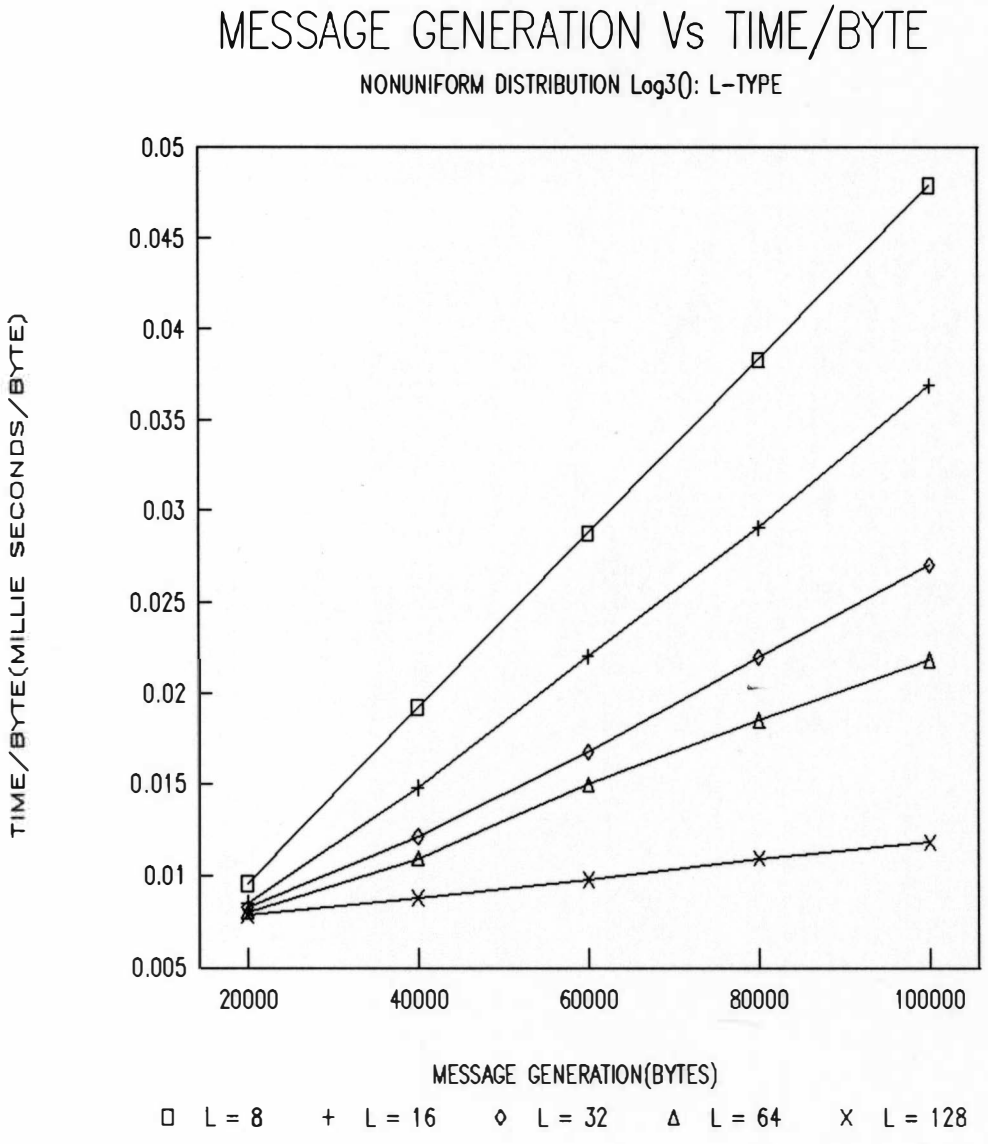


Figure 16. Relative Time per Byte Versus Message Generation Rate for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_3()$ Distribution.

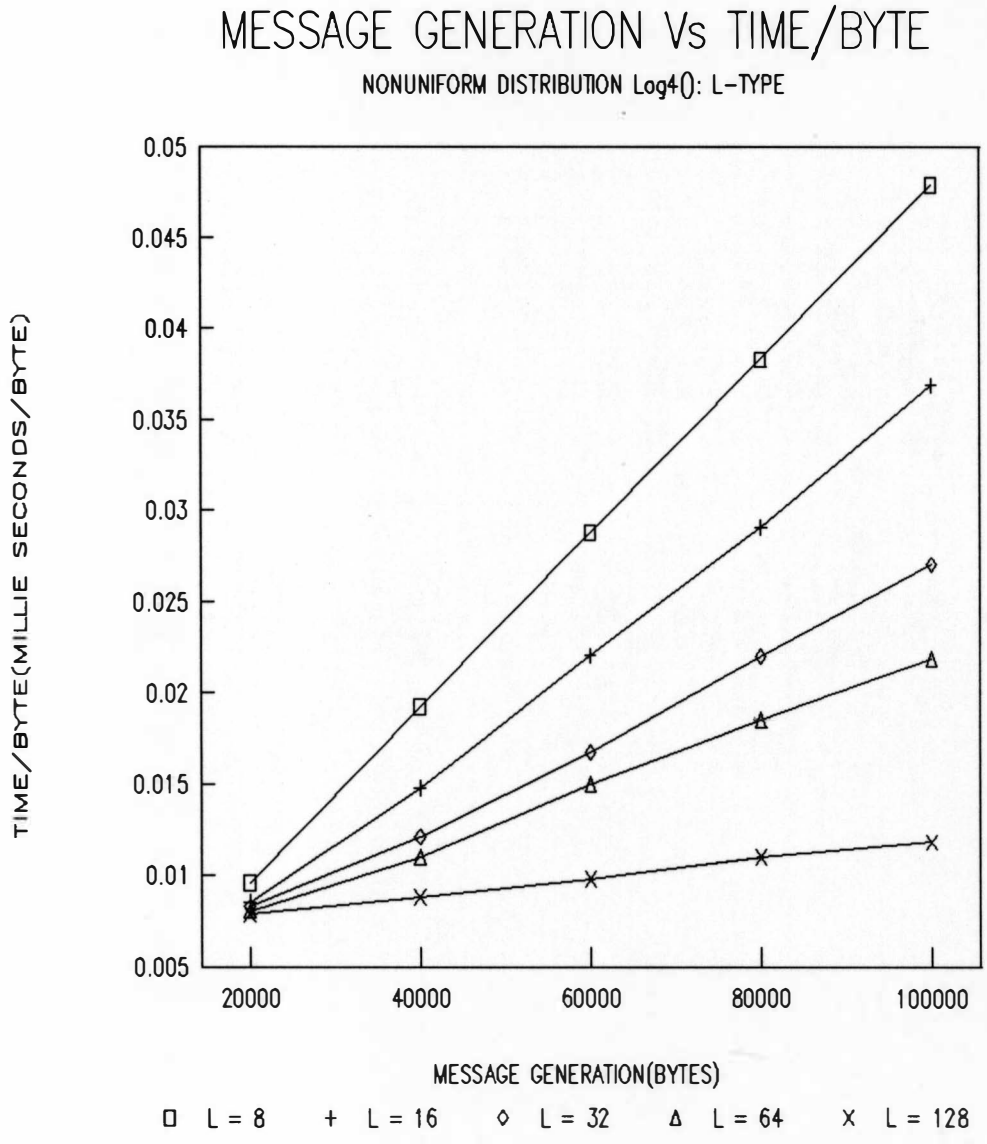


Figure 17. Relative Time per Byte Versus Message Generation Rate for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_4()$ Distribution.

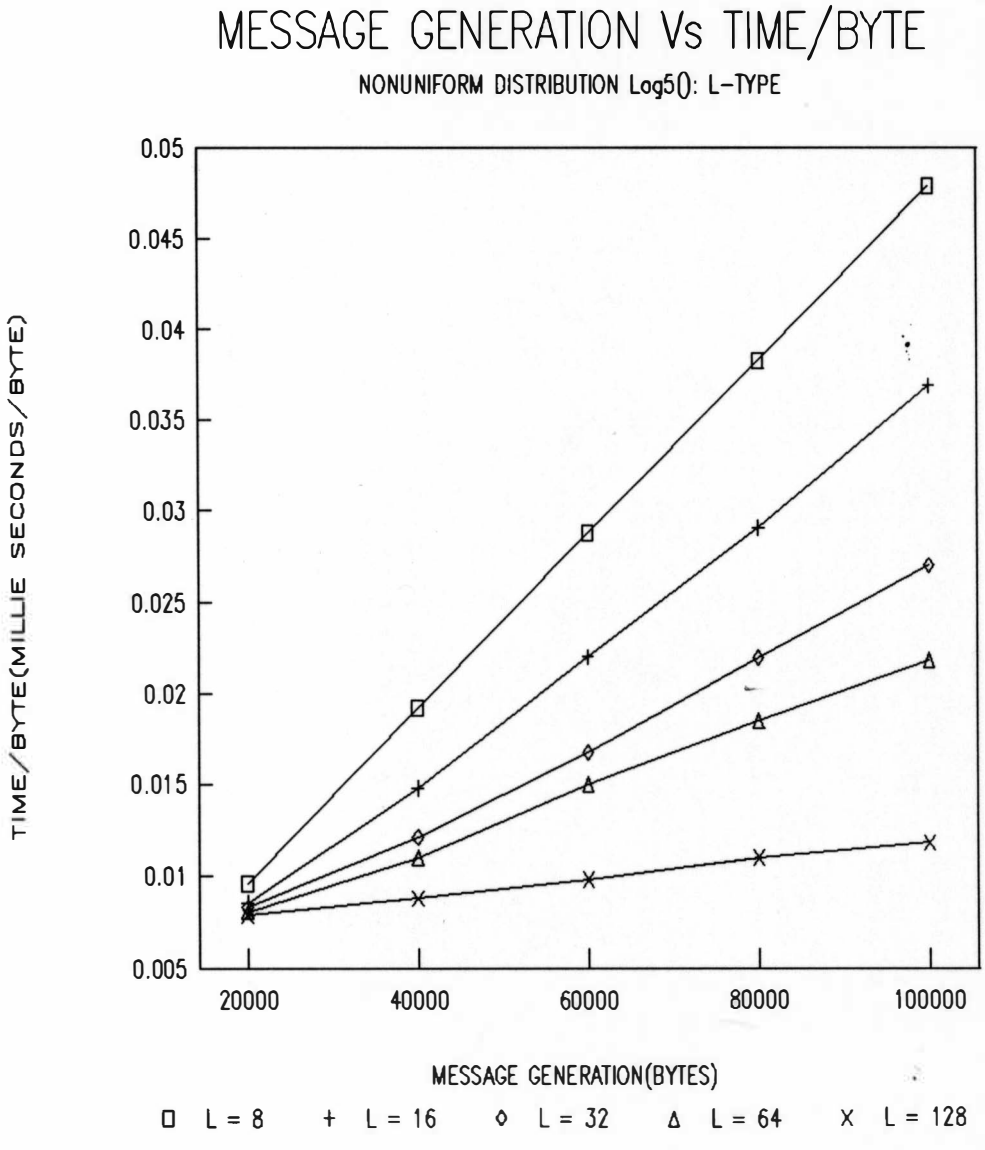


Figure 18. Relative Time per Byte Versus Message Generation Rate for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_5()$ Distribution.

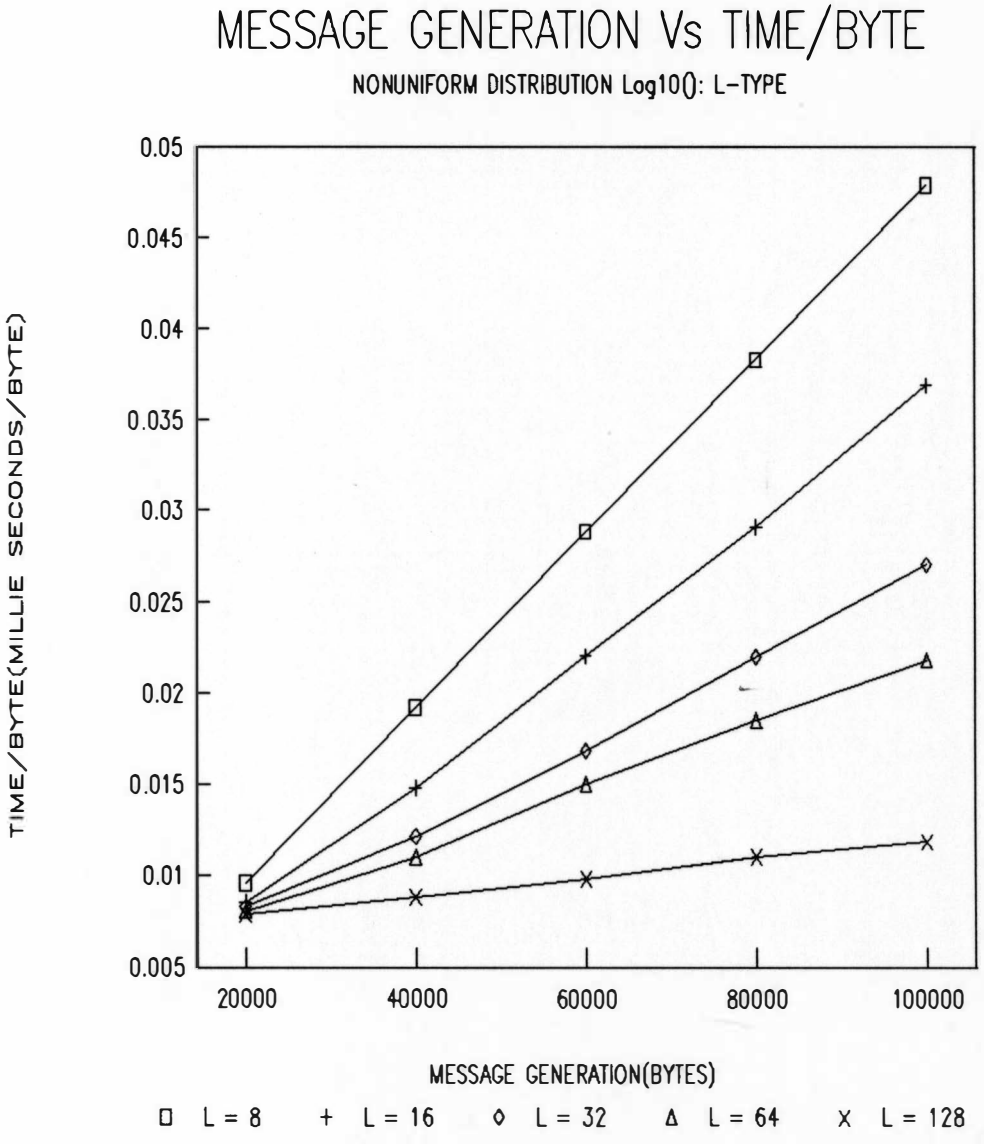


Figure 19. Relative Time per Byte Versus Message Generation Rate for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_{10}()$ Distribution.

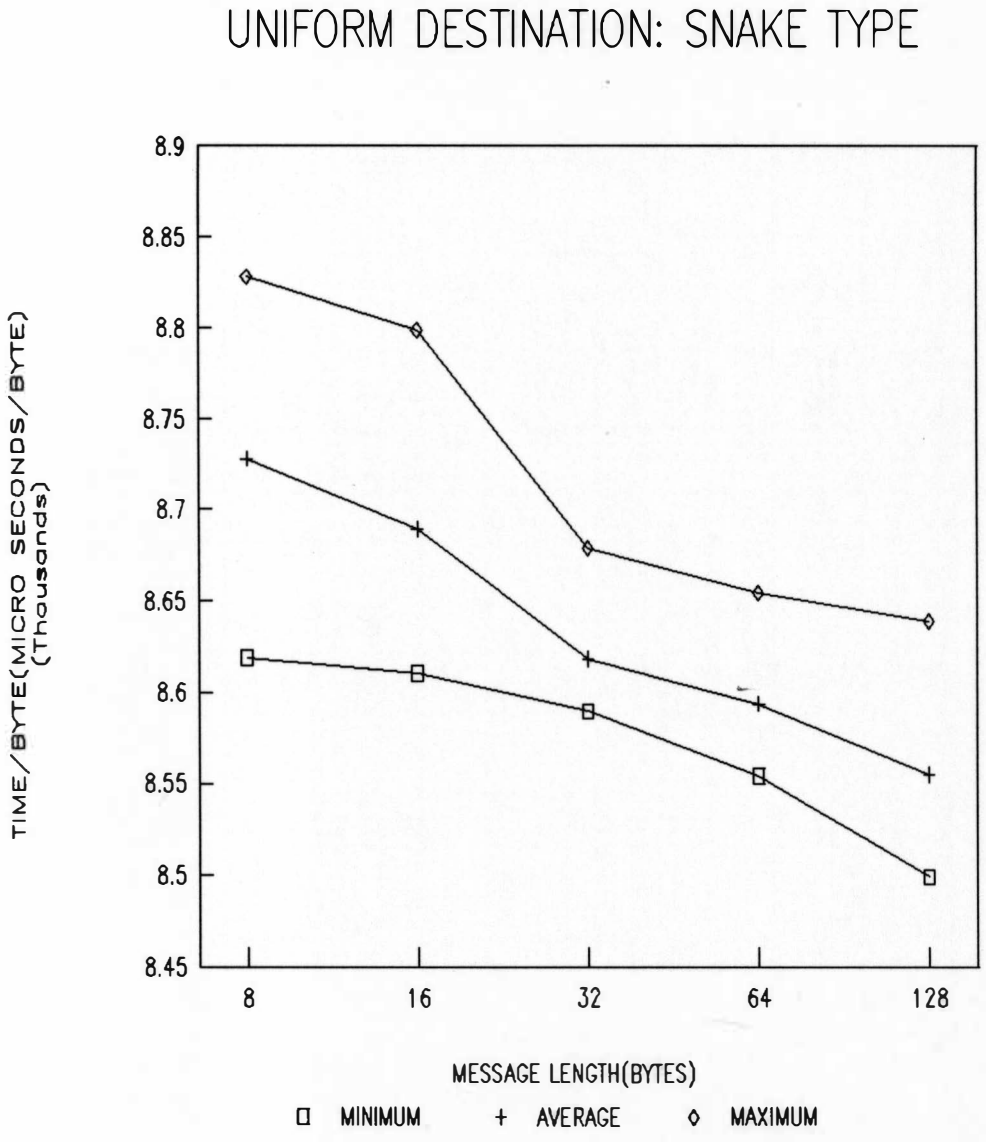


Figure 20. Relative Time per Byte Versus Message Length for Snake Type Transmission, for Uniform Distribution of Destination Processors.

associated with the round trip decreases exponentially. The decreases in relative time per byte is due to the fact that the message lengths L is broken into small packets and the setup time for communication for these packets overlaps with that of the transmission time of the previously transmitted packets so as time is saved in each of initialization processes of communication, thereby decreasing the round trip relative time per byte between any two processors in a give 2-dimensional toroidal mesh. This behavior of the nCUBE 2S system provides a hint at the hypothesis proposed by Voigt (Voigt, 1994), that for larger message length and increase in the number of packets shall bring the bandwidth to near channel speed. Figure 21 to Figure 25 are the plots for snake type transmission mode, but for non-uniform distribution of processors. In all these plots we can observe the same effect in the relative time per byte with respect to message length as what we have seen in the Figure 20.

An experiment was also performed for L type transmission, and uniform distribution of destination processors. By observing the graph, we can see the similarity to the results seen in the snake type transmission i.e., relative time per byte decreases as the message length tends to increase. The plot is shown in Figure 26. An experiment was carried for non-uniform distribution of processors with non-

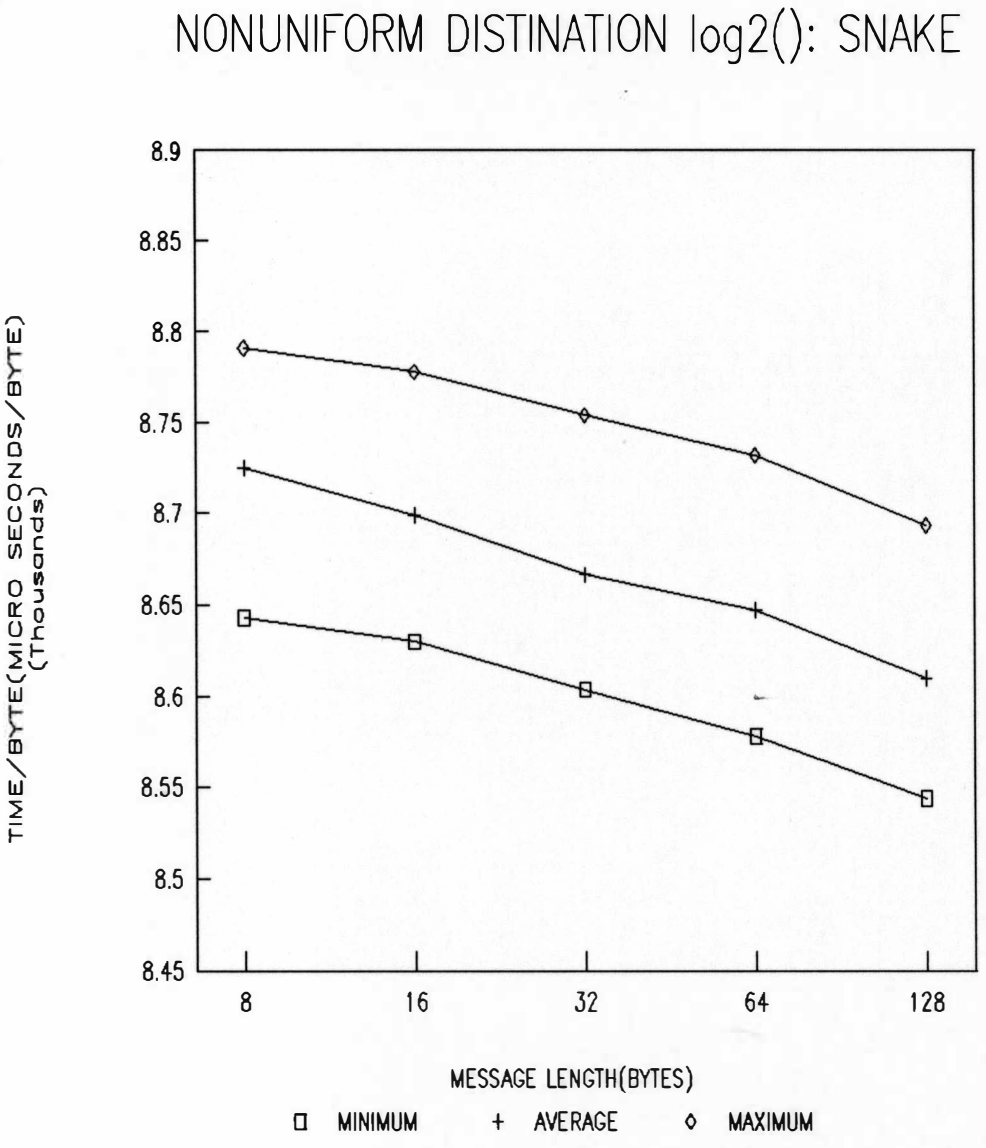


Figure 21. Relative Time per Byte Versus Message Length for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_2()$ Distribution.

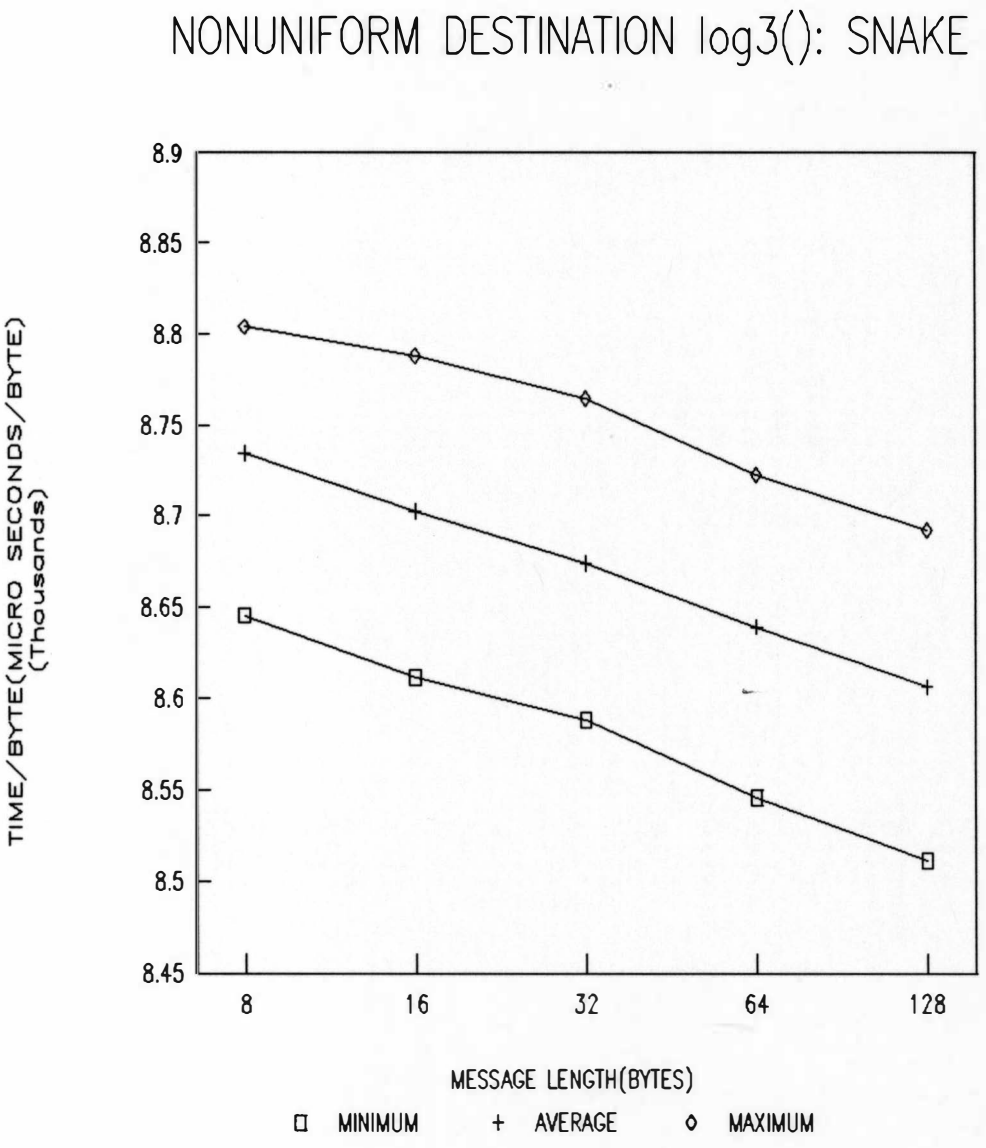


Figure 22. Relative Time per Byte Versus Message Length for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_3()$ Distribution.

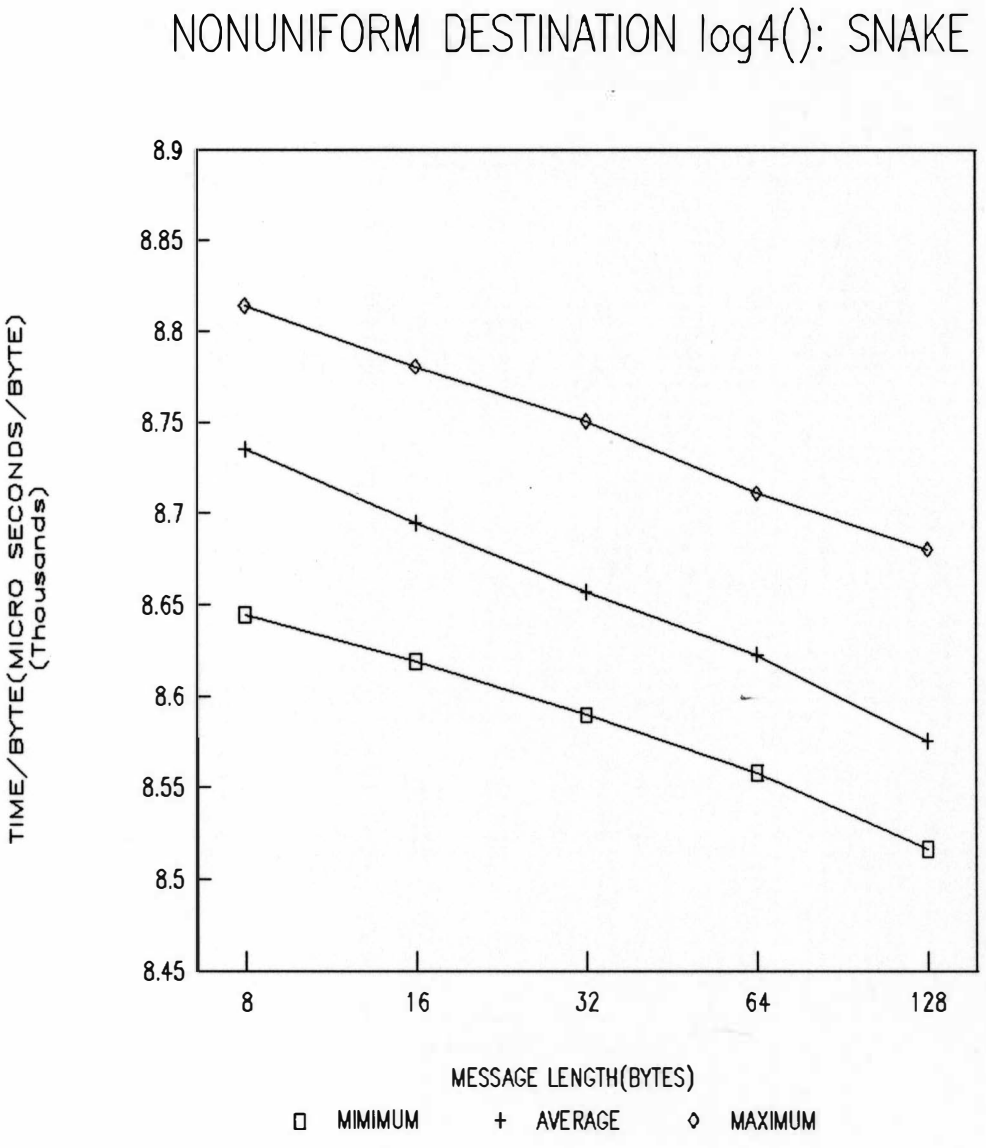


Figure 23. Relative Time per Byte Versus Message Length for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_4()$ Distribution.

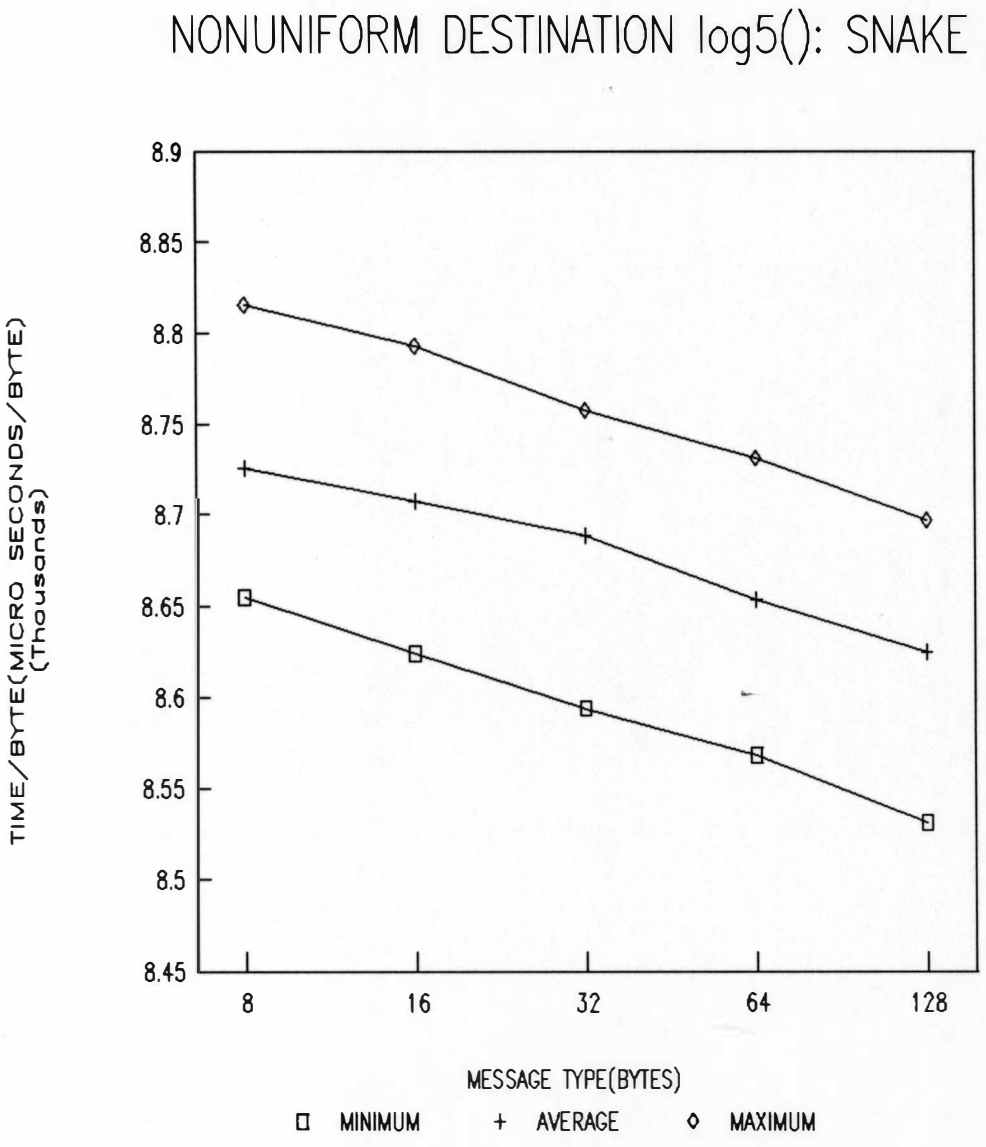


Figure 24. Relative Time per Byte Versus Message Length for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_5()$ Distribution.

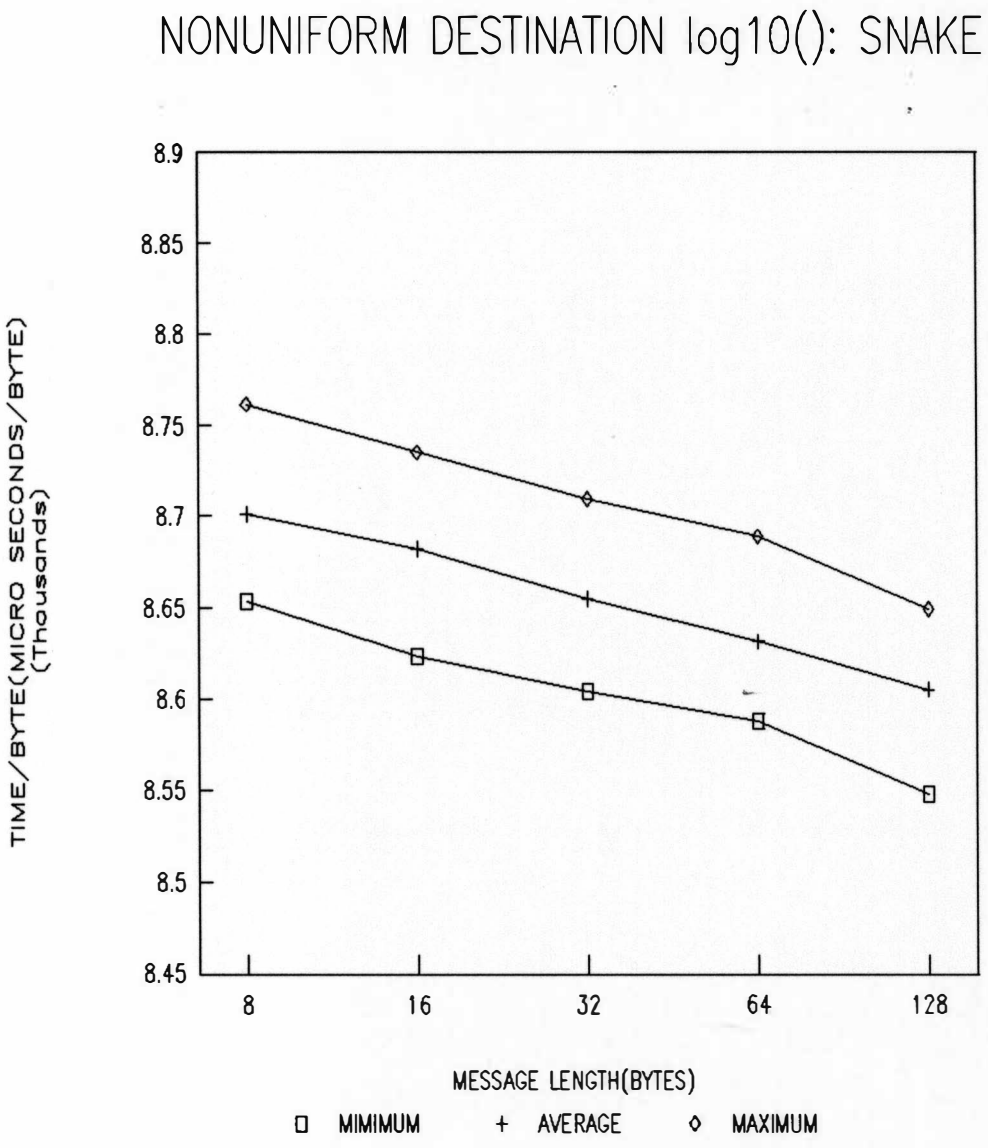


Figure 25. Relative Time per Byte Versus Message Length for Snake Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_{10}()$ Distribution.

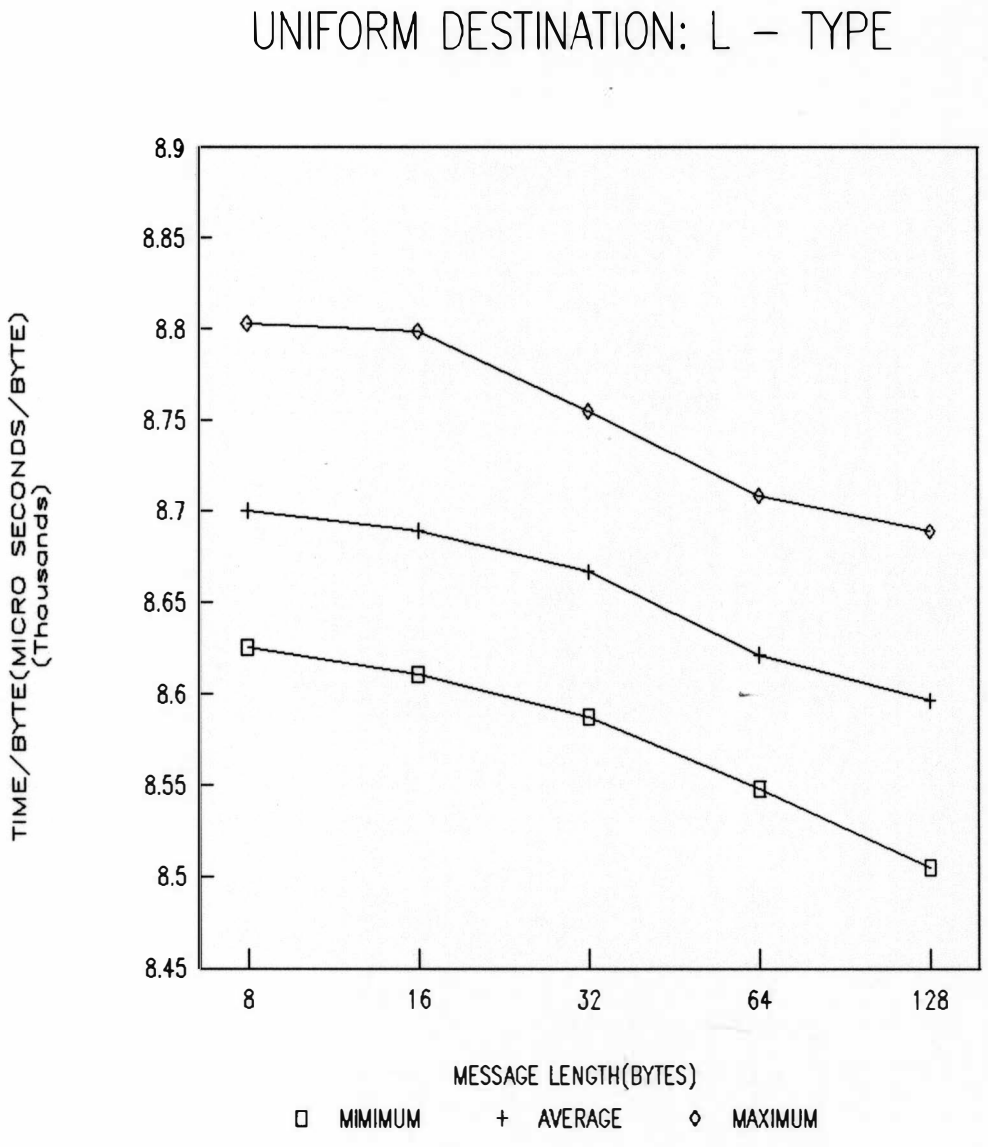


Figure 26. Relative Time per Byte Versus Message Length for L Type Transmission, for Uniform Distribution of Destination Processors.

uniformity of $\log_2()$, $\log_3()$, $\log_4()$, $\log_5()$, and $\log_{10}()$. Figure 27 to Figure 31 represent them. From these graphs we can once again observe the decrease in relative time per byte for increase in message length. Hence, in this experiment of time analysis, irrespective of mode of transmission, and distribution of the processors, the relative time per byte decreases with increase in message length. By closely observing the plots of snake type and L type modes of transmissions, we fine that snake type is more efficient than L type. The fact is because of that the snake type transmission mode has more overlapping time between the packets at the communication initialization time, where as the L type has a little less overlapping time.

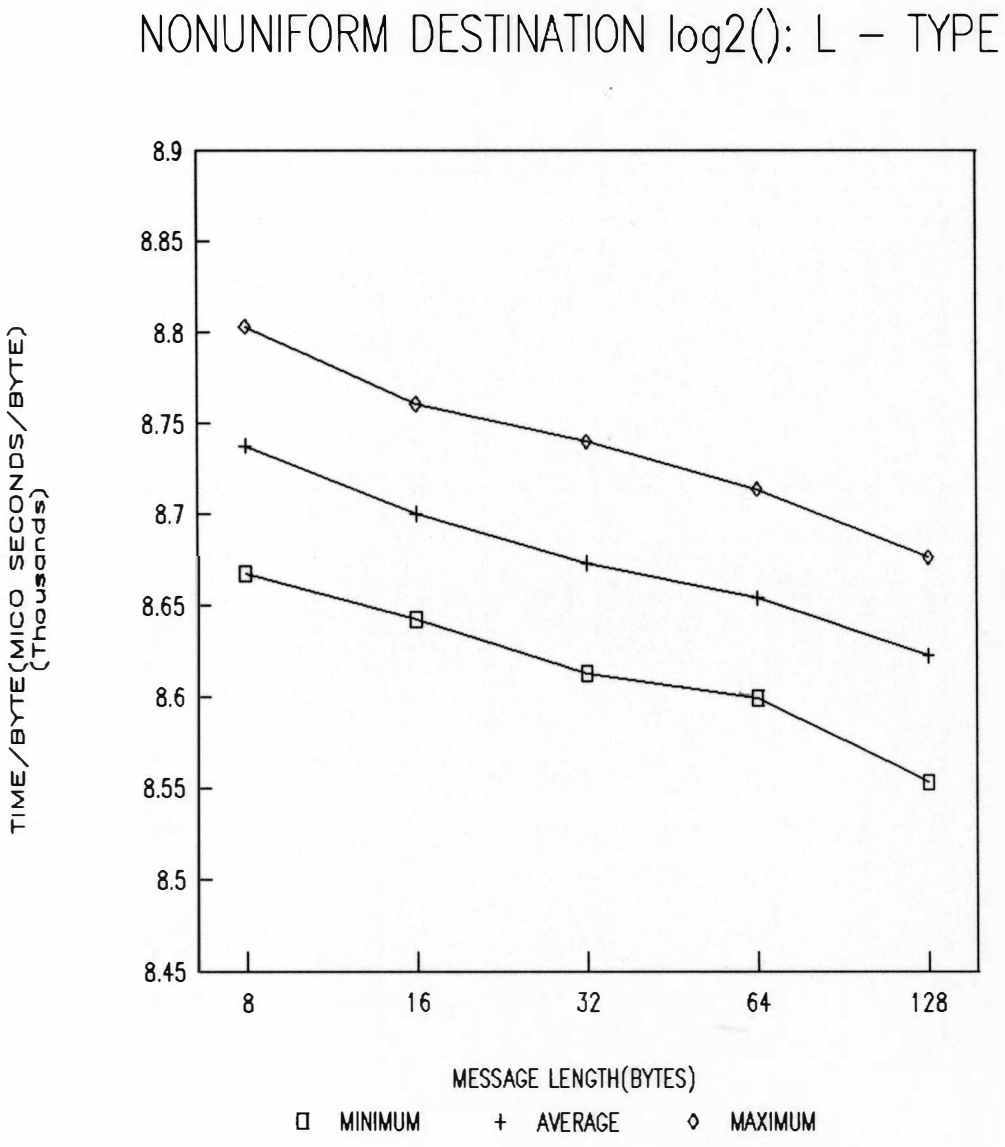


Figure 27. Relative Time per Byte Versus Message Length for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_2()$ Distribution.

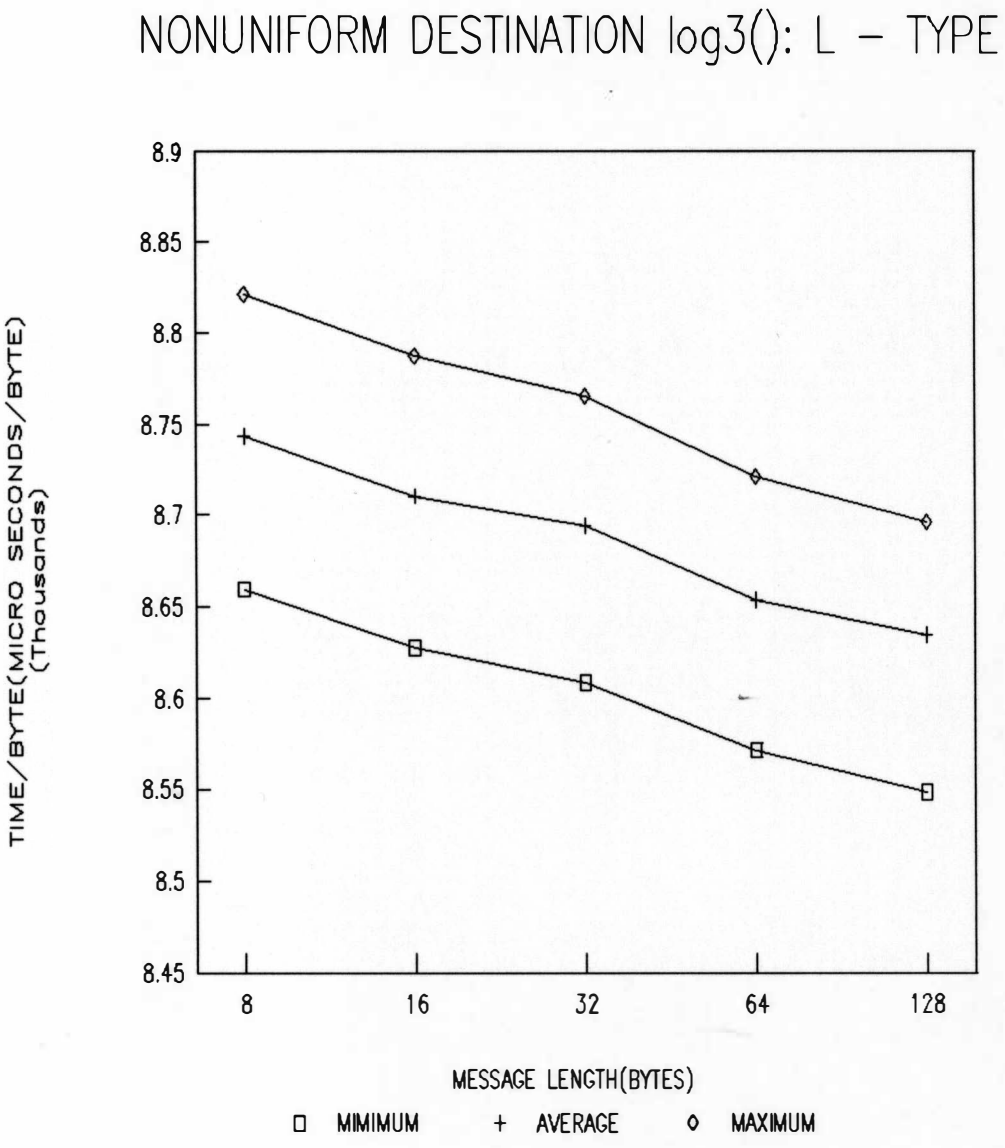


Figure 28. Relative Time per Byte Versus Message Length for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_3()$ Distribution.

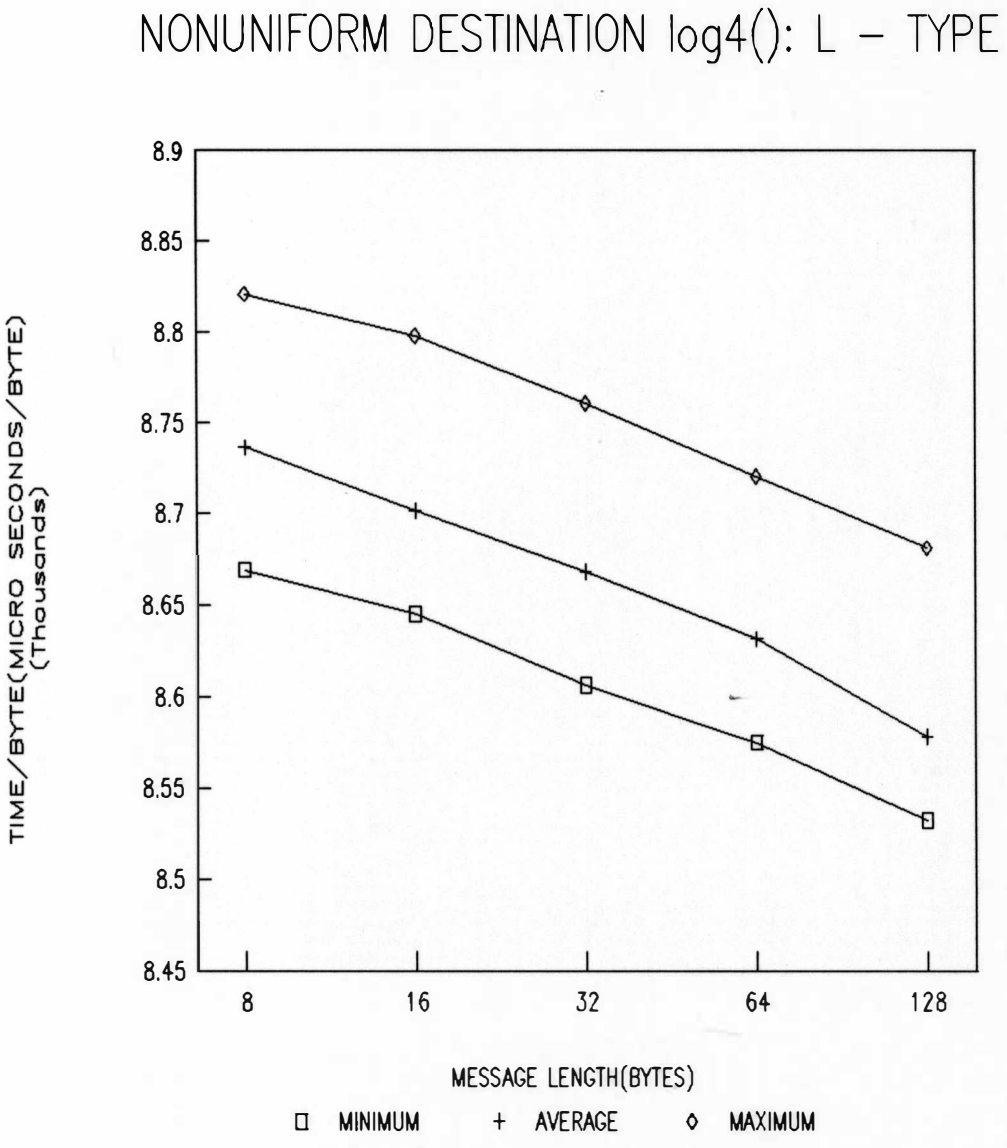


Figure 29. Relative Time per Byte Versus Message Length for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_4()$ Distribution.

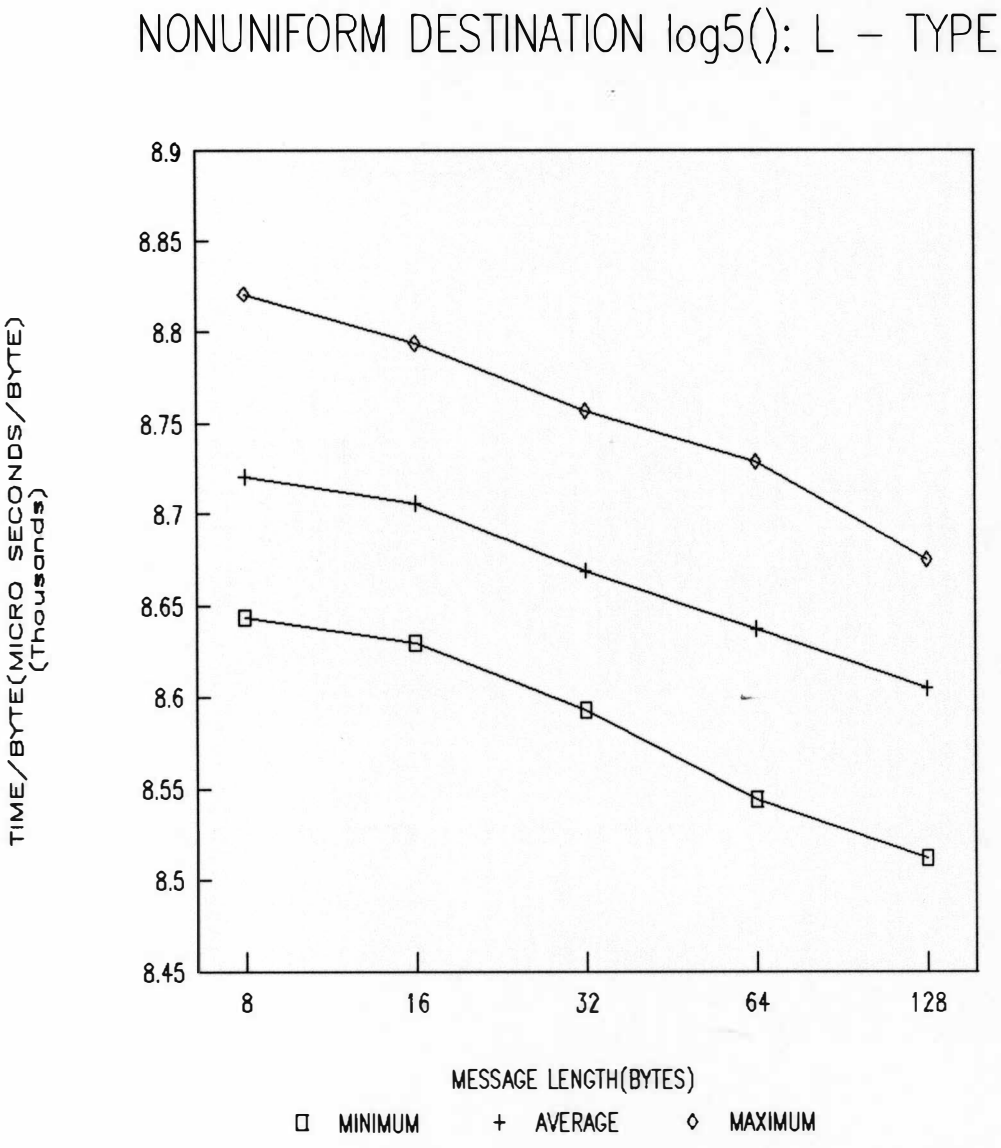


Figure 30. Relative Time per Byte Versus Message Length for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_5()$ Distribution.

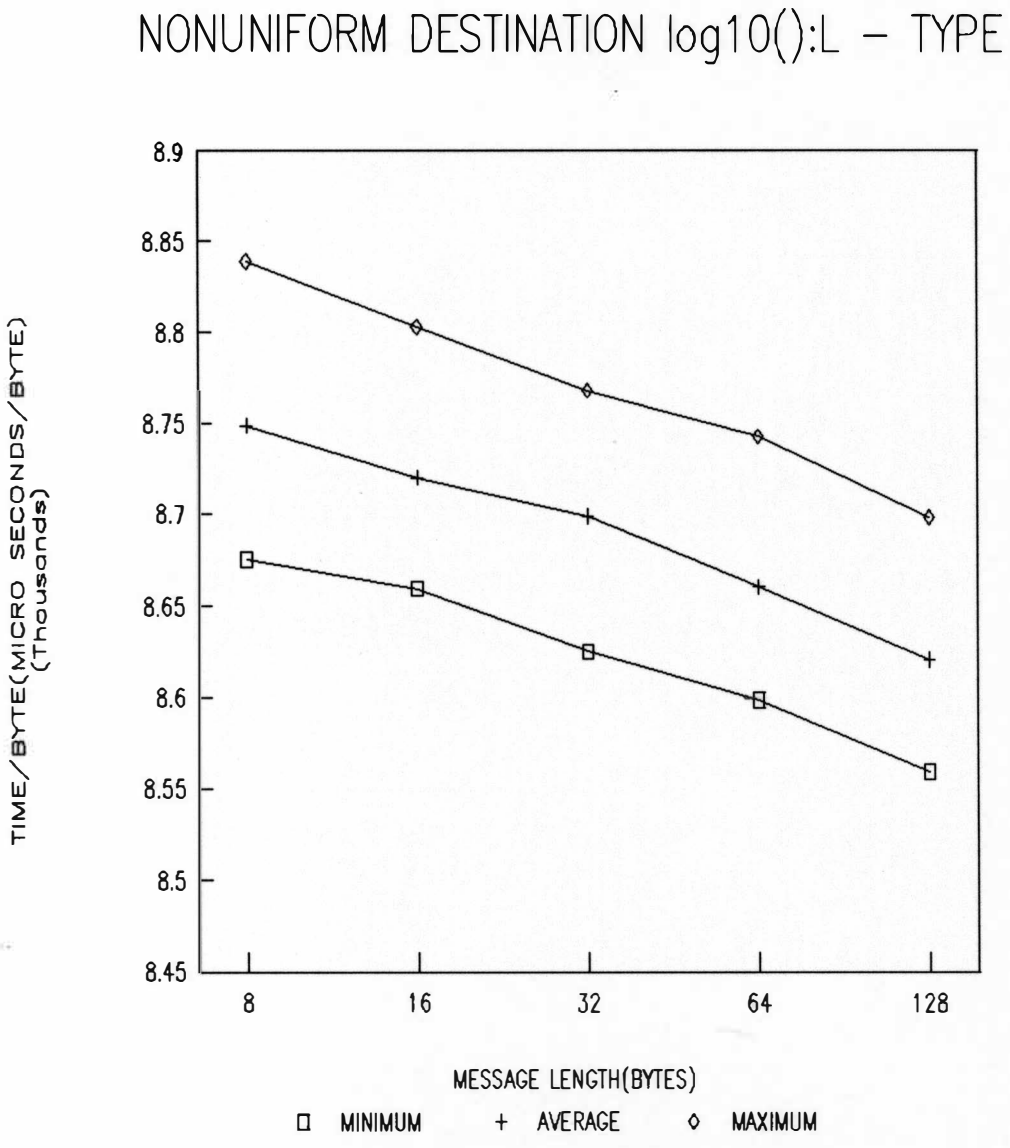


Figure 31. Relative Time per Byte Versus Message Length for L Type Transmission, for Non-Uniform Distribution of Destination Processors With $\log_{10}()$ Distribution.

CHAPTER IV

DESCRIPTION OF THE FEATURES OF THE MICRO-KERNEL

In this chapter we describe the features associated with the micro kernel that we used in order to perform the experiment in the previous chapter. As described in the previous chapter we implemented the micro kernel using the theoretical insight obtained from the statistical model. Where implementation of the statistical model was driven by cellular automata theory.

The functionality of the micro kernel is the root that it takes for the kernel to get implemented. Hence, let's look into to the functionality first. The kernel has to send and receive messages from any of the processors in the given network, as it is implemented with the help of cellular automata theory, the sending or receiving of messages can take place only through the neighboring processors, moreover the network that we use is a 2-dimensional toroidal mesh. Apart from above aspects the kernel has to look on to the communicational aspects, like buffer allocation for the messages to transmit or receive, allocate the data bus or channel, which is block until all the messages get transmitted, and then relieve the bus or channel for other usage. Along with these activities the

kernel also has to take control of the I/O operations associated with the computational processes.

As the test bed that we used was nCUBE 2S system, a class of hypercube architectural MIMD machine, and our theory requires 2-dimensional toroidal mesh, the kernel plays a role in getting the hypercube structure to get embedded into a 2-dimensional toraidal mesh. Moreover by the mean time setting up of the processes (getting ready to transmit messages) takes place once the destination processor is known. Buffers are set for the size of the message, and the channel for transmission are setup, which are blocked until the transmission gets completed. On the same time each and every processor tends to find its surrounding four neighbors. Depending up on the nearness the processor are with reference to the destination the messages get transmitted through them. The kernel also take care of the two modes of transmission i.e., the snake type and the L type mode of transmitting messages, and uniform and non-uniform way of distributing the destination processors. While these processes tends to initiate, a clock get triggered to calculate the time taken for the message for a round trip message travel.

Functions Used in Building of the Micro-Kernel

In this part of the chapter we focus up on the important library functions, that are used in building the

micro-kernel.

In the nCUBE parallel software environment (PSE) there is a status routine called *whoami*. It has the following arguments:

whoami(node, proc, host, dim)

where *dim* is the dimension of the allocated subcube, node the identification of the actual processor ranging from 0 to $2^{dim} - 1$. *host* is the identification of the front end which may be used to exchange data between the host and the subcube. *proc* is the process/processor ID. It is a 32-bit integer used by interprocessor communication routines for identifying a message's source or destination. The low 16 bits are a logical processor identification number (or node ID) that identifies an nCUBE 2S processor in the current subcube. When a subcube is allocated, nCX assigns each processor a number that uniquely identifies the processor's position in the subcube. These logical identification number range from zero to the number of processor in the subcube minus one. nCX assigns each process running on a processor a process ID, starting at 1. Ant time a subcube is allocated, the process ID of the first program loaded is set to 1. Bit 15 is a flag that is set to 0 if the processor is an nCUBE 2S processor in the processor array. The flag is set to 1 if the processor is any processor -an nCUBE 2S I/O processor or a processor on the host computer-

that is outside the processor array (Almasi & Gottlieb, 1989).

The nCUBE functions *nwright* an *nread* can be used for communication among processors. These routines accept the following arguments:

nwrite(buffer, length, dest, type)

nread(buffer, length, source, type)

Where *buffer* is the address of the first byte to be sent/received, length is the length of the message in bytes, *dest/source* the destination/source processor of the message to be sent/received and *type* is the type of the message. The message type is another identifying characteristic of a message. Both *whoami* and *nwrite/nread* functions were used in our kernel for identification of the processor as well as for the interprocessor communication.

Apart from these routines, one more important routine was used in building the kernel. As we mentioned before in order to perform the experiment the hypercube architecture of the nCUBE 2S system, we have to mapped the hypercube in to 2-dimensional toroidal mesh. This is performed by *nodetogrid* and *ngridtonode* routines. The arguments for these functions are as follow:

nodetogrid(proc, dim, ncoords, mask, GDIM, gdimsiz)

ngridtonode(dim, ncoords, mask, GDIM, gdimsiz, warp)

In the above arguments *ncoords* are the coordinates associated with the processor's position the given 2-dimensional

toroidal mesh, *mask* is a bit mask that specifies which hypercube neighbors the current node is to be communicate with. *GDIM* is the grid dimension, in our case it is 2, *gdimsiz* is the grid dimension size that is required for one to map the hypercube to 2-dimensional network. While *warp* is the setting to make warp around network, like the one we use. The above routines are used effectively to build the micro-kernel as small as possible. The complete code of the micro-kernel is place in the Appendix A.

CHAPTER V

CONCLUSIONS AND FURTHER WORK

In the first part of the of this chapter, we discuss the factors that affect the overall performance of our cellular load distribution schemes, taking an orthogonal view across the work presented in this thesis. By doing this as tie in all the results to provide a more complete picture. In the second section we make some suggestions for further work. The last section provides some concluding remarks. First, it is helpful to present the major contributions of this thesis. They are:

1. The study of systems' dynamic behavior under cellular load distribution strategy. We introduced a novel message distribution strategy, which distributes messages over a network of processors. We studied the relative speed of the message distribution and the factors that it depends on, namely message generation rate and message length.

2. The analysis of the performance of 'cellular' network as interconnection network for scalable system. A frequently noted problem of such network is their less than spectacular performance in system with message broadcasts. Using neighborhood based load distribution schemes, interacting processes are placed few links, or hops, apart.

With this type of strategy the message distribution shows simplicity in the network analysis.

Summary of Main Results

In Chapter II we introduces a statistical model based up on cellular automata theory and derived an expression for the performance of a 2-dimensional toroidal network. We also furnished the relationship between performance with respect to message length and message generation rate. In Chapter III we performed the experiment and tested the model on the test bed nCUBE 2S system. Comparison of the experimental results with the results of the statistical model, we can find a very good match on both the aspect of performance of the system with respect to message generation rate and message length. The are some variations in the performance between the experiment and theory, but the variations are about 0.03 to 0.06 micro seconds i.e., around 3 to 8 percent of difference are seen with experimental results on the uphill. These variations are caused by the re-transmission due to error correction that takes place while performing experiment, which the statistical model do not address because of consideration given to ideal transmission.

Further Work

This micro kernel talks about different modes of distributing messages and the efficient way of doing it. As

mentioned in the Chapter III, the micro kernels' method of distributing the messages looks more or like "fork" processes in UNIX system, which opens avenue for further research. This micro kernel can be used as a transporting shuttle for functional code modules, so as the modules can get evaluated at the destination processors and the evaluated modules can return back to the source processor for further computation of I/O operation. Apart from this it would be more advantageous if the messages are broken to small packets, and these packets can be transmitted for to get evaluated in parallel using different nodes, so as the speed can be improved by two ways, one through the way of gaining time by overlapping the transmission time at the initialization process as mentioned in the Chapter III, and the other by evaluating the small packet (code) in parallel. It would be an added advantage if the whole kernel is coded in any one of the functional languages. On the other hand, statistical model can be improved to study the characteristics of snake and L type modes of transmission if quantum mechanics principles like spin mechanism being introduced for every hop the message is going to take between source and destination processors. For example: A 1 hopspin for messages hoping to the right processor, -1 hopspin for the messages hoping to the left processor, 1 hopspin for messages hoping to the top processor, and -1 hopspin for messages hoping to the down processor can be introduced with

respect to the source processor. By doing so the total hopspin for a given source to destination transportation can be exactly estimated, which can provide a clear mathematical picture to determine the relationship between snake and L type modes of transmission.

Conclusions

The use of the cellular approach in the design and analysis of parallel processing systems was proposed and demonstrated. The power of this paradigm stems from four key factors. Firstly, cellular automata, whose theory underpins this approach, have a simple load interaction that allows mathematical analysis, and a complex global behavior that may be used to model larger physical systems. Secondly, cellular computing structures takes a radical step away from the traditional von Neumann architectures by incorporating processing capability in memory thereby avoiding the constraining processes-memory 'bottle-neck'. Thirdly, a significantly larger domain of important applications exists that are computer-intensive, and exhibit 'cellular' characteristics that allow them to be mapped almost directly onto cellular computing structures. Lastly, cellular structures are scalable, modularly extensible and be used of simple nodes and links, economical to extend.

The effectiveness of CLD depends on the characteristics of the load, the cellular architecture and the

parameters of the CLD scheme used. The key architectural factors are the network diameter, network size in number of processors, and the rate of increase span size with increase in span radius.

From both statistical and experimental analysis, we can conclude that irrespective of distribution of the destination processors or the mode of transmission, the relative time between messages decreases with respect to increase in message length, and increase in relative time between messages with respect to the message generation rate. By observing the plots made for both statistical and experimental results for both kinds of transmission modes, and both kinds of distribution types the results matches very well with each other. From the close match of the results between theory and experiment we can conclude that cellular automata theory can be used as an effective tool for modeling and development of massive parallel processing systems. By closely observing the plots of both snake type mode of transmission and L type model of transmission, we find that the snake type is more efficient than L type. This is due to a little less overlapping time takes place in the case of L type during the initialization processes of communication when compared with snake type mode of transmission of messages.

Thus with this axis of success with this micro kernel, cellular load distribution (CLD) strategy provides an

effective means of distributing load over large scalable network. CLD has low load distribution overheads directly due to simple and effective load distribution policies. Thus it is applicable at all levels of process grain and to a wide domain of task type (and structures), resulting in high system utilization levels and high speedup.

Appendix A

The Source Code of the Micro Kernel

DESCRIPTION OF THE CELLULAR MICRO-KERNEL CODE

The cellular micro-kernel code were built by effective use of nCUBE 2S library routines. Which facilitates the whole micro-kernel to be as small as possible.

Due to enormous usage of nCUBE 2S library routines, the header files associated with them are called. Then the initialization of the variables, buffer for the messages, clock for timings, and number of processes were set in so as the algorithms associated with snake and L type modes of transmission can be implemented. After the implementation of the snake and L type algorithms for both uniform and non-uniform distribution of destination of processors, a output file is opened to record the timings. Along with the closing of the output file, the program is terminated. The C code for both snake and L type modes transmissions of the cellular micro-kernel are furnished.

```
/*****************************************************************************************************/
/*                                                                                                   */
/*                                    CELLULAR MICRO KERNEL                                          */
/*                                          for                                                      */
/*      Message Distribution for L type Transmission Mode for Messages, for both Uniform             */
/*                and Non-Uniform Distribution of Destination of Processors.                         */
/*                                                                                                   */
/*****************************************************************************************************/


#include <stdio.h>
#include <ncube/npara_prt.h>  /* Header to Link nCUBE Library Routines */
#include <stdlib.h>
#include <nself.h>
#define GDIM 2                /* Defining the Grid Dimension           */

FILE *fp, *fpt;


main()
{

       int me, proc, host, nc_ncube, dest, node, new_node; /* Declaring the Charactres of the
                                                              Nodes                          */
        int length, type, flags, ravl, i, x, y;           /* Declaring the Characters of the
                                                              Messages                       */
        char *buf;                                         /* Allocation of Memory           */
        int from_me, from_dest, new_node;                  /* Declaring Source and Destination*/
        int ncoords1[GDIM];
         int ncoords[GDIM], mask[GDIM], gdimsiz[GDIM], warp[GDIM];
        int dcoords1[GDIM];                                /* Declaring Characters, which are*/
         int dcoords[GDIM];                                /* Used for Conversion of Grid to */
        int destcoords[GDIM];                              /* Node and vise-versa.           */
        int ngf, ngd, gnf;
        int ngf1, ngd1, gnf1;
        int start, elapse;                                 /* Setting for Timings            */
        double single;

         gdimsiz[0] = 8;                                   /* Declaring Grid Size            */
         gdimsiz[1] = 4;

        warp[0] = 1;                                       /* Getting Warp-Around            */
         warp[1] = 1;

        whoami(&me, &proc, &host, &nc_ncube);              /* Getting to know my Processor   */
        fflush(stdout);
        start = micclk();                                  /* Starting the Clock             */

        for(me = 0; me <= 31; me++)

        {

                for(i = 1; i <= 100000; i++)

                {

         type = 100;                                       /* Type of the Message            */
         length = 128; /* lrand48()>>28; */                /* Length of the Message          */
         flags = 0;
        x = lrand48()>>29; /* y = time()>>1; */            /* Declaring Uniform and          */
        y = log10(floor(x));                               /* Non-Uniform Distribution of
                                                              Destination Processors         */
        buf = (char*) malloc(length);                      /* Allotting the Memory           */

        ngf = nodetogrid(me, nc_ncube, ncoords, mask, GDIM, gdimsiz); /* Getting my Grid      */

        dcoords[0] = ncoords[0] + y;
        dcoords[1] = ncoords[1];
        dest = ngridtonode(nc_ncube, dcoords, mask, GDIM, gdimsiz, warp); /* Getting my
                                                                             Destination Node */
```

```
while(ncoords[0] != dcoords[0])                    /* Moving the Messages in L Mode
                                                      to the Destination              */

{

if(ncoords[0] != dcoords[0])

{
        if(ncoords[0] > dcoords[0])

                ncoords[0] = ncoords[0] - 1;

        else

                ncoords[0] = ncoords[0] + 1;
        new_node = ngridtonode(nc_ncube, ncoords, mask, GDIM, gdimsiz, warp);
         nwrite(buf, length, new_node, type, &flags);

}

}


while(ncoords[1] != dcoords[1])

{

if(ncoords[1] != dcoords[1])

{
        if(ncoords[1] > dcoords[1])

                ncoords[1] = ncoords[1] - 1;

        else

                ncoords[1] = ncoords[1] + 1;
         new_node = ngridtonode(nc_ncube, ncoords, mask, GDIM, gdimsiz, warp);
         nwrite(buf, length, new_node, type, &flags);

}

}

                if(ncoords[0] == dcoords[0] || ncoords[1] == dcoords[1]) /* Testing the
                                                                  Destination Coordinates */

                {

                        from_dest =  dest;                  /* Exchange of Coordinates*/
                        from_me = me;

ngf1 = nodetogrid(from_dest, nc_ncube, ncoords1, mask, GDIM, gdimsiz); /* Getting my  */
ngd1 = nodetogrid(from_me, nc_ncube, dcoords1, mask, GDIM, gdimsiz);   /* Grid for New
                                                                          Source and
                                                                          Destination*/

while(ncoords1[0] != dcoords1[0])

{

if(ncoords1[0] != dcoords1[0])

 {
        if(ncoords1[0] > dcoords1[0])

                ncoords1[0] = ncoords1[0] - 1;

        else

                ncoords1[0] = ncoords1[0] + 1;

        new_node1 = ngridtonode(nc_ncube, ncoords1, mask, GDIM, gdimsiz, warp);
        nwrite(buf, length, new_node1, type, &flags);

}

}
```

```
        while(ncoords1[1] != dcoords1[1])

        {

        if(ncoords1[1] != dcoords1[1])

        {
                if(ncoords1[1] > dcoords1[1])

                        ncoords1[1] = ncoords1[1] - 1;

                else

                        ncoords1[1] = ncoords1[1] + 1;
                new_node1 = ngridtonode(nc_ncube, ncoords1, mask, GDIM, gdimsiz, warp);
                nwrite(buf, length, new_node1, type, &flags);

        }

        }
                        }

                }


                elapse = micclk() - start;                      /* End the Clock                */
                single = (double)elapse/100000;
                fpt = fopen("1nu_5d_1xr.dat", "w");             /* Saving the Timings in a File */
                fprintf(fpt, "\n\n");
                fprintf(fpt, "Time to run  = %f microseconds\n", single);
                fclose(fpt);

        }

}
```

```
/*****************************************************************************************/
/*                                                                                       */
/*                                  CELLULAR MICRO KERNEL                                */
/*                                          for                                          */
/*    Message Distribution for SNAKE type Transmission mode of Messages, for both Uniform */
/*             and Non-Uniform Distribution of Destination of Processors.                */
/*                                                                                       */
/*****************************************************************************************/


#include <stdio.h>
#include <ncube/npara_prt.h> /* Header  to Link nCUBE Library Routines */
#include <stdlib.h>
#include <math.h>
#define GDIM 2              /* Defining the Grid Dimension */

FILE *fpt;


main()
{
        int new_node, new_node1, dest, from_dest, from_me; /* Declaring Source and Destination
                                                              Nodes                          */
        int me, proc, host, nc_ncube, i, x, y;             /* Declaring the Characters of the
                                                              Node                           */
        int length, type, flags, ravl;                     /* Declaring the Characters of the
                                                              Message                        */
        char *buf;                                         /* Allocation of Memory           */

        int ncoords[GDIM], mask[GDIM], gdimsiz[GDIM], warp[GDIM]; /* Declaring Characters,
                                                                     which are used for
                                                                     Conversion of Grid to Node
                                                                     and vise-versa.         */
        int dcoords[GDIM];                                 /* Destination Coordinate  */
       int ncoords1[GDIM];                                 /* Return Source Coordinate*/
       int dcoords1[GDIM];                                 /* Return Destination
                                                              Coordinate                     */
        int ngf, ngd, gnf;
       int ngf1,ngd1,gnf1;
        int start, elapse;                                 /* Setting for Timings     */
        double single;

        gdimsiz[0] = 8;                                    /* Declaring Grid Size     */
        gdimsiz[1] = 4;

        warp[0] = 1;                                       /* Getting Warp-Around     */
        warp[1] = 1;



        whoami(&me, &proc, &host, &nc_ncube);              /* Getting to know my
                                                              Processor                      */
        start = micclk();                                  /* Starting the Clock      */

       for(me = 0; me <= 31; me++)

        {

               for(i = 1; i <= 100000; i++)

               {

                length = 128; /*lrand48()>>28; */         /* Declaring the Message
                                                              Length                         */


                type = 100;                                /* Declaring the Message
                                                              Type                           */
                flags = 0;
               x = lrand48()>>29; /* y = time()>>1; */    /* Declaring Uniform and   */
               y = log10(floor(x));                        /* Non-Uniform Distribution*/
                                                           /* of Destination          */
                                                           /* Processors              */
```

```
buf = (char*) malloc(length);                          /* Allotting the Memory    */

ngf = nodetogrid(me, nc_ncube, ncoords, mask, GDIM, gdimsiz); /* Getting my Grid     */
dcoords[0] = ncoords[0] + y;
dcoords[1] = ncoords[1];

dest = ngridtonode(nc_ncube, dcoords, mask, GDIM, gdimsiz, warp); /* Getting my
                                                                     Destination Node*/

while(ncoords[0] != dcoords[0] || ncoords[1] != dcoords[1]) /* Moving the Messages in
                                                              Worm Mode to the
                                                              Destination          */
{

if(ncoords[0] != dcoords[0])

{
        if(ncoords[0] > dcoords[0])

                ncoords[0] = ncoords[0] - 1;

        else

                ncoords[0] = ncoords[0] + 1;
        new_node = ngridtonode(nc_ncube, ncoords, mask, GDIM, gdimsiz, warp);
        nwrite(buf, length, new_node, type, &flags);

}

 if(ncoords[1] != dcoords[1])

 {
        if(ncoords[1] > dcoords[1])

                ncoords[1] = ncoords[1] - 1;

        else

                ncoords[1] = ncoords[1] + 1;
        new_node = ngridtonode(nc_ncube, ncoords, mask, GDIM, gdimsiz, warp);
        nwrite(buf, length, new_node, type, &flags);

}

}
               if(ncoords[0] == dcoords[0] || ncoords[1] == dcoords[1]) /* Testing the
                                                                           Destination Coordinates*/
               {

                       from_dest =  dest;              /* Exchange of Coordinates   */
                       from_me = me;

ngf1 = nodetogrid(from_dest, nc_ncube, ncoords1, mask, GDIM, gdimsiz); /* Getting my  */
 ngd1 = nodetogrid(from_me, nc_ncube, dcoords1, mask, GDIM, gdimsiz);   /* Grid for New
                                                                           Source and
                                                                           Destination */

 while(ncoords1[0] != dcoords1[0] || ncoords1[1] != dcoords1[1])

{

 if(ncoords1[0] != dcoords1[0])
 {
        if(ncoords1[0] > dcoords1[0])

                ncoords1[0] = ncoords1[0] - 1;

        else

                ncoords1[0] = ncoords1[0] + 1;
        new_node1 = ngridtonode(nc_ncube, ncoords1, mask, GDIM, gdimsiz, warp);
        nwrite(buf, length, new_node1, type, &flags);

}

if(ncoords1[1] != dcoords1[1])

{
```

```
        if(ncoords1[1] > dcoords1[1])

                ncoords1[1] = ncoords1[1] - 1;

        else

                ncoords1[1] = ncoords1[1] + 1;
        new_node1 = ngridtonode(nc_ncube, ncoords1, mask, GDIM, gdimsiz, warp);
        nwrite(buf, length, new_node1, type, &flags);

    }

    }

            }
        }

        elapse = micclk() - start;                          /* End The Clock          */
         single = (double)elapse/100000;
         fpt = fopen("wnu_5d_1xr.dat", "w");                /* Saving the Timings in a
         fprintf(fpt, "\n");                                   File                    */
         fprintf(fpt,"Time to run  = %f microseconds\n", single);
         fclose(fpt);


    }

}
```

REFERENCES

Abramowitz . M. & Stegun, I. A. (1970). Hand Book of Mathematical Functions. New York: Dover Publications, Inc.

Almasi, G. S., & Gottlieb, A. (1989). Highly parallel Computing. Readwood City: Benjamin/Cummings Publication.

Bunt, R. B., & Murphy, J. M. (1984, August). The Measurement of Locality and the Behavior of Programs. BCS Computer Journal, 27(3), 238-245.

Burks, A. W. (1970). von Neumann's Self-Reproducing Automata. Urbana: University of Illinois Press.

Codd, E. F., (1968). Cellular Automata. New York: Academic Press.

DeBenedictis, E., & del Rosario, J. M. (1992). nCUBE Parallel I/O Software. IEEE Eleventh Annual International Phoenix Conference on Computer Communication, 117-24.

Demongeot, J., Goles, E., & Tchuente, M. (1985). Dynamical Systems and Cellular Automata. New York: Academic Press.

Edwards, A. L. (1971). Probability and Statistics. New York: Holt, Rinehard and Windston, Inc.

Fahlman, S. E., Hinton, G. E., & Sejnowski, T. J. (1983). Massively Parallel Architectures for AI, NETL, THISTLE, and BOLTZMANN Machines. Proceedings of the National Conference on AI.109-13.

Gajski, D. D., & Peir, J. K. (1985, June). Essential Issues in Multiprocessor System. IEEE Computer, 18(6), 9-27.

Ghosal, D., Tripathi, S. K., Bhuyan, L . N., & Jiang, H. (1989, September). Analysis of Computation-Communications Issues in Dynamic Dataflow Architectures. 16th Ann. Int. Conf. on Computer Architecture, 325-33.

Goodeve, D., & Taylor, R. (1992). CHaOS: A Scaleable

Cellular Kernel for MIMD Tasking. Parallel and Distributed Computing in Engineering Systems, 9-17.

Gray, H. L. & Odell, P. L. (1970). Probability for Practicing Engineers. New York: Barnes and Noble, Inc.

Greenwood, G. (1995). Private communication.

Gubbala, N., & Singh, C. (1993). Reliability Analysis of Interconnected Power System Using Monte-Carlo Simulation in nCUBE Parallel Computer. Proceedings of the American Power Conference, 2, 1609-14.

Harary, F., Hayes, J. P., & Wu, H. J. (1988). A Survey of the Theory of Hypercube Graphs. Comput. Math. Applic. 15. 277-289.

Haynes, L. S., Lau, R. L., Siewiorek, D. P., & Mizell, D. W. (1982, January). A Survey of Highly Parallel Computing. Computer, 9-24.

Hillis, W. D. (1984). The Connection Machine: A Computer Architecture Based on Cellular Automata. Physica D, 10, 213-28.

Hillis, W. D. (1985). Connection Machine, MIT Press.

Hwang, K., & Briggs, F. (1985). Computer Architecture and Parallel Processing. New York: McGraw-Hill.

Kleinrock, L. (1975). Queuing Systems. New York: J. Wiley and Sons.

Klietz, A. E., Malevsky, A. V., & Chin-Purcell, K. (1994, August/September). Mix-and-Match High Performance Computing. IEEE Potentials. 6-10.

Lindermayer, A. (1968). Mathematical Models for Cellular Interactions in Development. Part I & II, Jour. Theor. Biology, 18.

Little, D. C. (1961). A proof of the Queuing Formula: $L = \lambda W$, Operations Research, 9, 383-87.

Macharia, G. M. (1990). Cellular Load Distribution: Dynamic Load Balancing in Scalable Multicomputers. York, UK: University of York Press.

Mead, C., & Conway, L. (1980). Introduction to VLSI Systems. New York: Addison Wesley.

Morgan, G. (1987). The Cellular Modelling of Fault-Tolerant Multicomputers. York, UK: University of York Press.

Neumann, J. V. (1956). Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components. Automata Studies. Princeton: Princeton University Press.

Palmer, J. F. (1988). The NCUBE Family of High Performance Parallel Computer Systems. Third Conference on Hypercube Concurrent Computers and Applications, 1, 847-51.

Sargeant, J. (1987). Load Balancing, Locality & Parallelism Control in Fine-Grain Parallel Machines. Manchester, UK: University of Manchester Press.

Schaffer, G. (1995). MPP UNIX enhancement for OLYTP Applications. Proceedings of the Twenty-Eighth Hawaii International Conferences on System, Vol. 1, 124-33.

Schmidt, S. C., Dick, R. D., Forbes, J. W., & Tasker, D. G. (1992). Hydrocode Development on the nCUBE and Connection Machine Hypercube. Shock Compression of Condensed Matter - 1991 Proceedings of the American Physical Society to Special Topic Conference. 289-92.

Shannon, C. E., & McCarthy, J. (1956), Automata Studies, Princeton: Princeton University Press.

Taylor, R. (1995). Private communication.

Toffoli, T. (1984). CAM: A High-Performance Cellular Automaton Machine, Physica, 10D, 195-204.

Vitanyo, P, M, B. (1987). Locality, Communication, and Interconnect Length in Multicomputers. Amsterdam: University of Amsterdam Press.

Voigt, M. S. (1994). Efficient Parallel Communication with the nCUBE 2S Processor, Parallel Computing, 20, 509-530.

Waltz, D, L. (1987). Application of the Connection Machine. IEEE Computer, 20(1), 85-97.

Wendler, K. (1981). Models to Describe those Features of Cellular Computer Nets Which are Relevant to the Operating System, Computer Supplementum, 3, 193-203.

Wolfram, S. (1986). Theory and Applications of Cellular

Automata. World Scientific.

Yalamanchili, S., & Aggarwal, J. K. (1987). A Characterization and Analysis of Parallel Processor Interconnection Networks. IEEE Trans. on Computers, C-36(6), 680-691.