

AD-A194 361

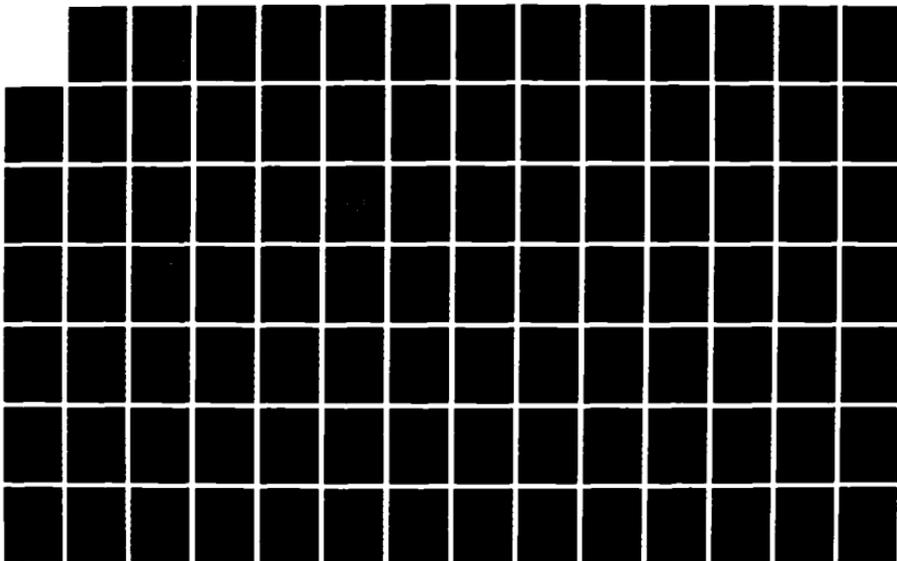
A PERFORMANCE STUDY OF THE HYPERCUBE ARCHITECTURE(U)  
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL  
OF ENGINEERING C A LAMANA JUN 88 AFIT/OCS/ENG/88J-1

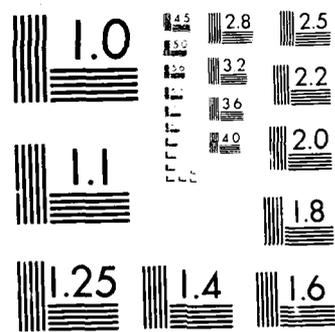
1/2

UNCLASSIFIED

F/B 12/6

NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS 1963 A

0

DTIC FILE COPY

AFIT/GCS/ENG/88J-1

AD-A194 361

A PERFORMANCE STUDY OF THE  
HYPERCUBE ARCHITECTURE

THESIS

Catherine Anne Lamanna  
Captain, USA

AFIT/GCS/ENG/88J-1

DTIC  
SELECTE  
JUN 23 1988  
D

Approved for public release; distribution unlimited

81

AFIT/GCS/ENG/88J-1

A PERFORMANCE STUDY OF THE HYPERCUBE  
ARCHITECTURE

THESIS

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology  
Air University  
In Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science in Computer Engineering

Catherine Anne Lamanna, B.S.  
Captain, USA

June 1988

Approved for public release; distribution unlimited



## *Table of Contents*

	Page
Acknowledgments . . . . .	ii
Table of Contents . . . . .	iii
List of Figures . . . . .	vi
List of Tables . . . . .	vii
Abstract . . . . .	viii
I. Introduction . . . . .	1
Background . . . . .	1
Hypercube . . . . .	2
Statement of the Problem . . . . .	4
Scope and Limitations . . . . .	4
Approach . . . . .	5
Overview . . . . .	9
II. Literature Review . . . . .	10
General . . . . .	10
Performance Models . . . . .	11
Hypercube Hardware Characteristics . . . . .	13
Summary . . . . .	13
III. Research Method . . . . .	15
Overview . . . . .	15
Benchmark . . . . .	15

	Page
Benchmark Program . . . . .	15
Benchmark Experiment . . . . .	18
Data Collection . . . . .	18
Simulation Model . . . . .	19
Model Components . . . . .	19
Model Construction . . . . .	22
Model Validation . . . . .	27
Experiment Design . . . . .	28
Levels of the Control Variables . . . . .	29
Data Collection . . . . .	30
Summary . . . . .	31
IV. Results . . . . .	32
Evaluation of the First Research Hypothesis . . . . .	32
Benchmark Results . . . . .	34
Computation to Communications Ratio . . . . .	37
Evaluation of the Second Research Hypothesis . . . . .	39
Evaluation of the Third Research Hypothesis . . . . .	44
V. Conclusions . . . . .	45
General . . . . .	45
Research Results . . . . .	45
Benchmarking Versus Simulation . . . . .	45
Impact of Workload on Speedup . . . . .	46
Suggestions for Further Research . . . . .	47
Summary . . . . .	47
A. Benchmark Program . . . . .	48

	Page
B. SLAM II Source Code . . . . .	54
C. Benchmark Results . . . . .	70
D. Model Validation . . . . .	80
Simulation Results . . . . .	81
Paired-Difference t Test . . . . .	82
E. Simulation Results . . . . .	83
Bibliography . . . . .	103
Vita . . . . .	105

*List of Figures*

Figure		Page
1.	Hypercube Topology . . . . .	3
2.	Research Approach . . . . .	7
3.	Flow of Benchmark Program . . . . .	17
4.	Plot of Message Transmission Times . . . . .	21
5.	Concurrent Activities . . . . .	23
6.	Simulation Flow of Events (Part A) . . . . .	25
7.	Simulation Flow of Events (Part B) . . . . .	26
8.	Comparison of Benchmark and Simulation Times . . . . .	34
9.	Plot of Total, Computational and Message Processing Times . . . . .	35
10.	Comparison of Models . . . . .	37
11.	Plot of Speedup for Four Computational Loads . . . . .	38
12.	Plot of Speedup for Total Computational Load of 1000 ms . . . . .	41
13.	Plot of Speedup for Total Computational Load of 20,000 ms . . . . .	42
14.	Comparison of Actual and Predicted Values . . . . .	43

*List of Tables*

Table		Page
1.	Research Hypotheses . . . . .	4
2.	Benchmark Levels of Control . . . . .	19
3.	Uniprocessor Loads Used for Validation . . . . .	28
4.	Control Variables . . . . .	29
5.	Experiment Design . . . . .	30
6.	Benchmark Versus Simulation Times . . . . .	33
7.	Simulation Results . . . . .	40

*Abstract*

This study investigated the relationship between workload characteristics and process speedup. There were two goals: the first was to determine the functional relationship between workload characteristics and speedup, and the second was to show how simulation could be used to determine such a relationship. The hypercube implementation used in this study is a packet-switched network with predetermined routing. Message processing has precedence, so nodes are interrupted during task processing.

In this study, three independent variables were controlled: total computational workload, number of nodes and the message traffic load. The workload was assumed to be balanced across the nodes. A benchmark program was executed on an actual hypercube and the results were used to validate a discrete event simulation model of hypercube processing. Using the simulation, an experiment was designed to control the total computational load over two levels, the number of nodes over five levels and the message traffic load over four levels to determine their individual and interactive effects on process speedup.

Regression analysis was used to estimate the functional relationship between the three independent variables and process speedup. The results show that a complex relationship exists between workload characteristics and cube size. As more nodes are added, the computational time decreases, but at the same time, the communications overhead increases such that the speedup will eventually begin to decrease. The point where speedup starts to decline is dependent upon both the computational and message traffic workload. Finally, this research presented an alternative methodology for performance analysis which is more flexible than the traditional methods. Furthermore, this methodology can be extended to study other architectures.

# A PERFORMANCE STUDY OF THE HYPERCUBE ARCHITECTURE

## *I. Introduction*

### *Background*

Parallel processing has become an attractive solution for applications that require a large amount of computation in a short period of time. Since the computational requirement of a single problem is distributed among several processors, there must be some mechanism for communication between processors. The manner in which a multiprocessor handles communication between processors classifies it as either a loosely-coupled or tightly-coupled machine. The former communicates via a common memory, whereas the latter uses a message-transfer system. The communications required of a process directly affect the time needed to complete the process.

The speedup that can be achieved by a parallel computer with  $n$  identical processors working concurrently on a single problem is at most  $n$  times faster than a single processor. In practice, the speedup is much less, since some processors are idle at a given time because of conflicts over memory accesses or communication paths ... [HW81]

A speedup of  $n$  is achievable only if a multiprocessor is operating at peak performance. During peak performance, the processors are doing only useful work; no work is redundant and no extra instructions are executed. That is, the parallelization does not require more instructions than a uniprocessor would require using the same algorithm. An ideal speedup is impaired by several factors which include:

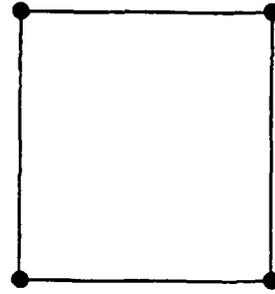
- interprocessor communication.
- processor synchronization.
- one or more idle processors.
- wasted effort, and
- processing required for system control and scheduling [ST87] .

### *Hypercube*

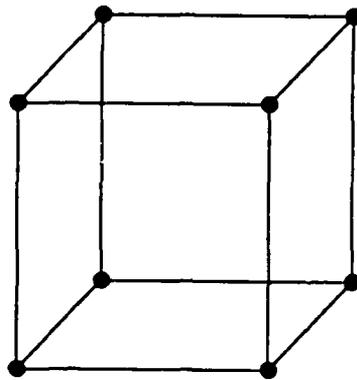
The hypercube is a loosely-coupled multiprocessor. Its interconnection network is a binary  $n$ -cube, so it connects  $N = 2^n$  nodes where each node is a processor with its local memory and  $n$  is the dimension of the cube. Figure 1 shows the logical topology of cubes of one-, two-, three-, and four-dimensions. The vertices represent nodes and the edges represent point-to-point full-duplex communication paths. In a cube of dimension  $n$ , every node is directly connected to  $n$  other nodes that are called nearest neighbors. Every node is capable of communicating with every other node, but messages sent to non-nearest neighbors must pass through intermediate nodes. The performance of certain algorithms implemented on hypercubic architectures has been measured. Felten *et al.* recorded efficiencies over 90% when solving the traveling salesman problem [FE85]. Gutzmann found that when he implemented a sorting algorithm on a hypercube, a larger dimension (two-dimensional) cube "actually performed worse than...smaller ones for a given list size. The reason for this is that the communications overhead ... is usually larger than direct processing costs..." [GU87]. This particular algorithm also required processors to drop out and become idle before the sorting was completed. The first example represents results that are encouraging while the other is discouraging. The specific concern is, how much faster can a process complete if more processors are added to the job and what characteristics of the workload affect speedup performance.



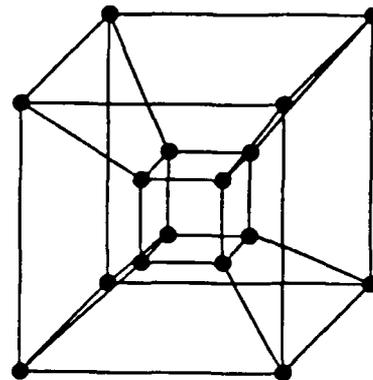
a) One-Dimension



b) Two-Dimensions



c) Three-Dimensions



d) Four-Dimensions

Figure 1. Hypercube Topology

Table 1. Research Hypotheses

- H1: There is no difference in hypercube speedup explained by choice of benchmark or a simulation model of a hypercube architecture for a controlled workload.
- H2: There is no difference in hypercube speedup explained by total computational workload, number of processors, message traffic load and their interactions.
- H3: There is no difference in hypercube speedup explained by the distribution of burst times of individual processors where the total workload on each node is the same.

#### *Statement of the Problem*

The speedup of a process that can be run in parallel on a hypercube is affected by both the computational load placed on each processor and the message traffic load between processors. The purpose of this thesis is to present a functional model for determining the impact of these factors on speedup. This model allows for the description of the structure of a parallelized algorithm and prediction of speedup over various sizes of the hypercube. The research hypotheses are listed in Table 1.

#### *Scope and Limitations*

The computational load and the message traffic load must be characterized to determine the effects of these two factors on the speedup of a process running in parallel. Since unbalancing the load over processors severely degrades speedup [MO87], the total computational load is assumed to be evenly distributed and can run concurrently.

The speedup experienced by using multiple processors is determined by relative execution time on a single processor. Although a particular process may run faster on another uniprocessor, the comparative measure must be relative to a processor of the same type.

The goodness of a particular parallelized algorithm is not considered. The model is a means by which the performance of an algorithm on the hypercube can be estimated given the computational and message traffic load characteristics. The total execution time does not account for time needed to download programs or data to the hypercube. Although this time is significant, it is more of a function of the algorithm and hypercube implementation than it is architectural performance. Ni *et al.* have dealt with elimination of this bottleneck during algorithm design [NI87a].

#### *Approach*

The goal of this thesis is to determine the effects of the computational load, the message traffic load and their interaction on the speedup of a process for several dimensions of the hypercube. To meet this goal, an experiment is designed so that the variables can be controlled independently of each other. Data is collected from benchmarking and simulation, and the hypotheses in Table 1 are tested by statistical means. Finally, the results are analyzed and interpreted. Figure 2 shows the steps which are summarized below and described in detail in Chapter III.

Step 1: Identify the independent variables.

- (a) Three primary independent variables are total computational workload, number of processors and message traffic load. The total computational workload is quantified as the time for a single processor to complete the process. The workload placed on a single processor does not include any communication overhead to slow down total execution time. The second independent variable, the number of processors, is a quantity that can be controlled directly. The third variable, the message traffic load, is quantified by the total number of messages generated during the process.
- (b) A secondary independent variable is the processors' burst times between transmission of messages. The burst times are characterized as being approximately the same for all bursts or as being completely random. Since the computational workload is distributed evenly across the processors, this variable must be quantified as a binary variable; either the burst times are approximately the same or they are not.

Step 2: Benchmark the effects of the primary independent variables on an actual hypercube such as Intel's Personal Super Computer (iPSC).

- (a) A matrix multiplication algorithm is a good candidate to implement as a benchmark program. Using this algorithm, the computational workload can be varied independently of the message traffic load by iterative recalculation of matrix elements.
- (b) The processors' computational times and message generation and receipt times are measured across hypercubes of dimension 1 through 5 for several different computational workloads.

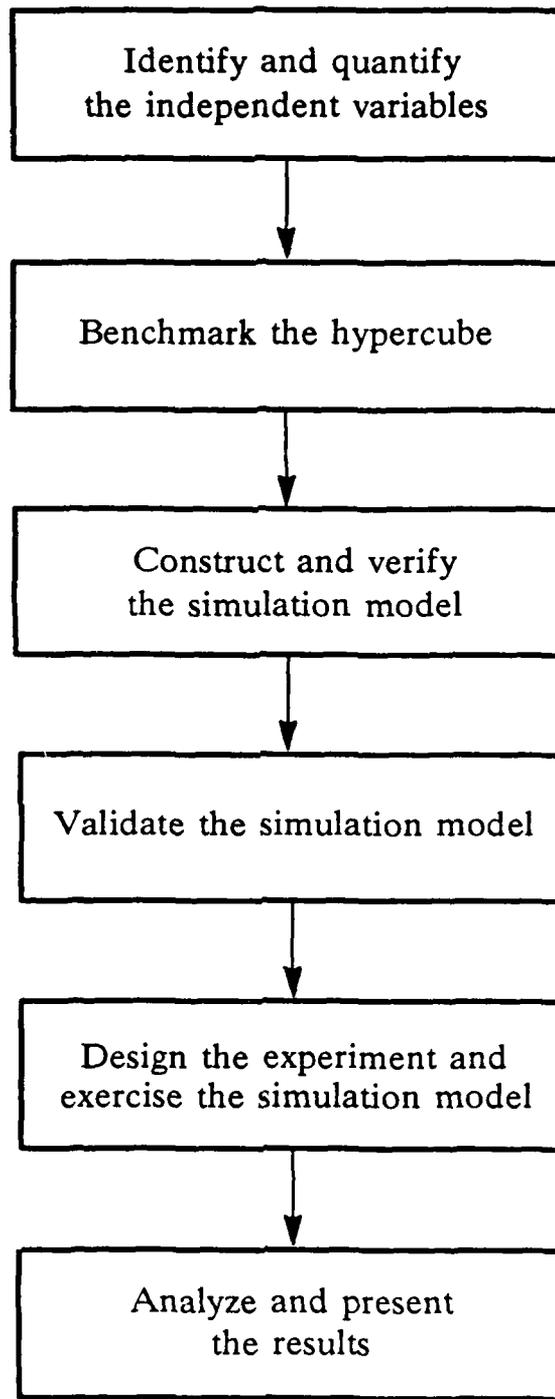


Figure 2. Research Approach

Step 3: Construct and verify a simulation model using discrete event simulation. A simulation of a process running in parallel facilitates control of the independent variables and measurement of the effects due to workload characteristics without having to design and implement actual workloads for the hypercube. The model is constructed under the conditions of: an evenly balanced workload, each processor executes the same number of bursts, and messages are generated between bursts. Model construction includes determining specific times that are used in the model:

- (a) The iPSC's interprocessor communication times are required to model message passing between nodes. To capture the portion of communications that is concurrent with processing, it is necessary to decompose the message transmission time into its components and to estimate a time for each component.
- (b) The amount of additional processing time in the matrix multiplication program resulting from checking the receipt of messages is required to calibrate the simulation model to the matrix multiplication algorithm.

Step 4: Validate the simulation model by comparing the results of the model to the results of the benchmark and ensuring through statistical means that there is no difference between the two.

Step 5: Design an experiment that exercises the simulation model in which the total workload, number of processors, message traffic load and burst duration are all varied independently of one another to determine their main and interactive effects on the speedup of a process run in parallel.

Step 6: Analyze and present the results. The relationships between the independent variables are presented by testing the second and third research hypotheses listed in Table 1. Under the conditions of the model, the relationships can be

used to understand and predict relative speedups as a function of the independent variables and draw conclusions about the nature of speedup phenomena.

### *Overview*

Chapter II gives a summary of current knowledge. Although there is an abundance of literature on multiprocessing, Chapter II includes literature that is directly related to this thesis. Chapter III describes the development of the benchmark and the simulation model and presents the experimental design. Chapter IV discusses the results of the benchmark and the simulation. Chapter V summarizes the results and suggests how the results may be used to predict performance of algorithms on the hypercube.

## *II. Literature Review*

### *General*

There has been a considerable amount of research concerning the performance of multiprocessors. Much of it involves evaluating the performance of a particular algorithm on a given architecture. The performances measured are a function of the algorithm design, the architecture and the interaction between the two. That is, how well the algorithm is mapped to the architecture. Evaluating the performance of an architecture is difficult because it cannot be divorced from the algorithm used to evaluate it; after all, the algorithm dictates what kind of workload will be placed on the hardware.

The performance of a parallelized algorithm is sensitive to the coupling between processors. Loosely-coupled machines are more efficient when processor interaction is low, whereas a tightly-coupled machines are more tolerant of interaction [HW84]. LeBlanc conducted an experiment to find the tradeoffs between two configurations of the BBN Butterfly Parallel Processor. This machine can support a shared-memory as well as a message-passing capability. LeBlanc's case study concluded that matching the application and the computational model was more important than the model itself [LE86].

A constraining factor governing the processor interaction that a loosely-coupled system can tolerate and still have reasonable efficiency is the performance of the interconnection network. The saturation point for communications is characteristic of the interconnection structure [ST87]. The hypercube is an open-ended architecture unlike shared-memory and bus architectures which are limited in their expansion. Also, since it is a directly connected network, processes exhibiting affinity can be assigned to nearest-neighbor processors to take advantage of communication locality [SE85]. Nicol and Willard took advantage of this property in their algorithm and

found linear speedups since there was no contention for communication resources. Communication costs for transmitting data between partitions were independent of total communications load [N187b]. Another feature of the hypercube is its adaptability to other communication topologies for different applications. Wiley showed that a four-dimensional cube could be used to emulate a two- or three-dimensional mesh, a ring or a tree structure [W187].

### *Performance Models*

A basic model for total execution time of processes on multiprocessors or distributed systems has been given by both Indurkha *et al.* and Stone. This model assumes the processors are connected with a bus. If a process consists of  $M$  tasks each requiring  $R$  time units to complete and the communication costs is  $C$  time units, then using a two-processor system, the total execution time,  $T_e$ , is given by

$$T_e = R \max(M - k, k) + C(M - k)k \quad (1)$$

where  $k$  is the number of tasks assigned to one of the processors. The optimal assignment policy of tasks which minimizes this function is to distribute the tasks evenly between the processors if  $M/2 < R/C$ , otherwise all tasks are assigned to one processor [IN86a] [ST87].

This basic model was extended to  $N$  processors, where the total execution time is given by

$$\begin{aligned} T_e &= R \max(k_i) + \frac{C}{2} \sum_{i=1}^N k_i (M - k_i) \\ &= R \max(k_i) + \frac{C}{2} M^2 - \sum_{i=1}^N k_i^2 \end{aligned} \quad (2)$$

where  $k_i$  is the number of tasks assigned to the  $i$ th processor. The optimal assignment policy is no different from the one given above [IN86a] [ST87].

Under the optimal assignment policy, the cost of evenly distributing the tasks over  $N$  processors is

$$T_c = \frac{RM}{N} + \frac{CM^2}{2} - \frac{CM^2}{2N} \quad (3)$$

and the speedup is given by

$$\begin{aligned} S &= \frac{RM}{\frac{RM}{N} + \frac{CM^2}{2} - \frac{CM^2}{2N}} \\ &= \frac{\frac{RN}{C}}{\frac{R}{C} + \frac{M(N-1)}{2}} \end{aligned} \quad (4)$$

This means that for small  $N$  and  $M$  and large  $R/C$ , the speedup depends more on  $N$ , but when  $N$  gets large enough, the speedup is proportional to  $R/CM$  and does not depend on the number of processors [ST87].

An assumption made under the previous models is that every task must communicate with every other remotely assigned task. If the assumption is changed so that information sent to a processor is distributed to all its resident tasks, then a linear communications cost model is appropriate. This means that communications cost is proportional to the number of processors instead of the number of tasks. The total execution time for this model is given by

$$T_t = R \max(k_i) + C \cdot N \quad (5)$$

An even distribution of tasks produces the best time, so the first term would become  $RM/N$ . Execution time,  $T_t$ , is minimum when

$$N = \sqrt{\frac{RM}{C}} \quad (6)$$

The effective parallelism is reduced because of communication costs [ST87]. Although communications are not completely serial, Chapter IV will show that a linear communications cost model is also applicable to the hypercube when every processor is required to communicate once with every other processor.

### *Hypercube Hardware Characteristics*

Reed and Grunwald benchmarked the iPSC's transmission time between nearest-neighbor processors. Assuming that a message consists of single packet, they modeled the transmission time as

$$S_{CL} = t_l + Lt_c \quad (7)$$

where  $t_l$  is the communications latency,  $L$  is the length of the message in bytes and  $t_c$  is the transmission time per byte. Using a least squares fit to their data, they found that  $t_l = 1.7$  milliseconds and  $t_c = 2.8$  microseconds. Their evaluation did not include intermediate node hand-off time for messages that require multiple links [RE87]. Moore's thesis [MO87] and this thesis jointly duplicated this benchmark. The results are compared in Chapter III.

Wiley claimed that a hypercube can support a computational to communications ratio of 10 to 1. The ratio is measured as the rate the nodes execute instructions to the communications bandwidth [WI87]. For the iPSC, the nodes execute 1 million of instructions per second and the channel bandwidth is 10 megabits per second. This implies that if the 10 to 1 ratio is not followed, communications channels will become saturated. Wiley further stated that programmers need to keep this ratio in mind when writing software for the hypercube [WI87]. Another computation to communication ratio is derived in Chapter IV which may be more amenable to programmers for determining the best sized hypercube for their application.

### *Summary*

The literature strongly suggests that the performance of a multiprocessor is a result, in part, of the algorithm chosen to run on it. Clearly, the architectural performance is dependent on the workload placed on the architecture. Stone's [ST87] performance models use variables that describe the workload. That is,  $RM$  describes the total computational load and  $(M^2 - \sum_{i=1}^N k_i^2)$  describes the message traffic load. His models assume that the process can be partitioned in  $M$  tasks exhibiting the

same behavior. The model presented in Chapter IV also uses variables that describe the workload but it is extended to account for heterogeneous tasks.

### *III. Research Method*

#### *Overview*

The purpose of this thesis is to determine the effects of the computational load, message traffic load and their interaction on the speedup of a parallellized process over five dimensions of the hypercube. In Chapter I, the independent variables that describe the workload were identified and quantified. To determine their effects, an actual hypercube was benchmarked using a workload that could be controlled. The benchmark program placed a homogeneous workload on each processor. To preclude the design and implementation of various workloads for the hypercube so that all the independent variables could be controlled, a simulation model of hypercube processing was constructed, verified and validated with the benchmark. To study the effects of the workload characteristics, an experiment was designed and the simulation model was exercised. Regression was then used to determine the functional relationships of the independent variables. These results are presented in Chapter IV.

#### *Benchmark*

Intel's Personal Super Computer (iPSC) was benchmarked using a parallellized matrix multiplication program. A useful program was selected instead of a kernel that merely placed a workload on the hypercube so that results of the program could be verified for correctness. A useful program also gives an indication of the overhead associated with implementation details, such as, how a process verifies receipt of its messages. Since that overhead is a function of the particular implementation, it is not studied in this thesis, but it cannot be ignored nonetheless.

*Benchmark Program* The benchmark program squares an  $8 \times 8$  matrix where each node computes a submatrix, passes its results to every other node and verifies

that is has received the results from every other node. A copy of the program is downloaded to the hypercube by the host, which is a front-end processor that serves as a link between the hypercube and its users. Figure 3 is a high-level flowchart representation of the nodes' program which is listed in Appendix A and explained below.

1. Each node opens communications channels to the host and the cube. These are logical communications channels and are used in the routines that handle message transfers. The nodes receive a message from the host telling them how many processors are active in the cube.
2. Each node computes the indices of the first and last elements of its submatrix based on its node identification number and the number of active nodes in the cube. Each node will compute  $64/N$  elements where  $N$  is the number of active processors.
3. Each node marks its start time.
4. Each node computes its submatrix results  $x$  number of times. Variations in  $x$  are what allow the computational workload to be controlled.
5. Each node marks the time it completes its computation.
6. Each node sends its results to every other active node in the cube. The nodes' messages are uniquely identified by a different type. The nodes' identification number is assigned by the program as the message type.
7. At each node, a flag for each message type is stored in a vector. The flag indicates whether or not the message has been received. The type identifies where the message was originated, thus indicating where in the results matrix the incoming data needs to go and a buffer pointer can be set to that location. The vector of flags is scanned and the type of an unreceived message is identified.

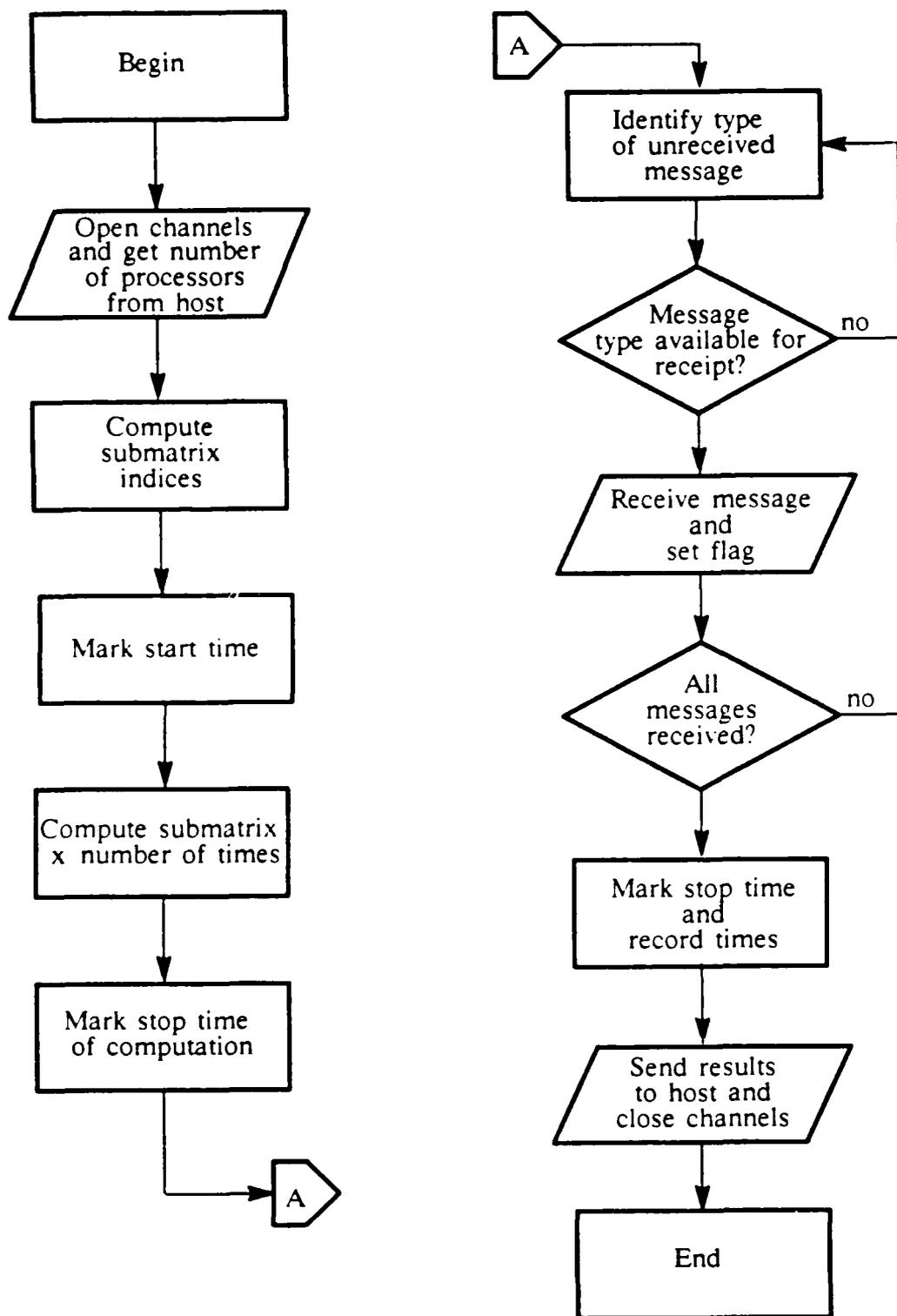


Figure 3. Flow of Benchmark Program

8. Nodes check to see if the identified message type is available for receipt.
9. If the message type is available, the nodes receive the message directly into the appropriate place in the results matrix.
10. Each node uses the number of messages it expects to receive as a flag. If all the messages are not received, then another message type is identified.
11. When each node has received all of its messages, it marks the stop time then records its times to the system log.
12. Each node sends its results matrix to the host then closes its communication channels.

*Benchmark Experiment* The experiment was designed such that the total computational load is varied over six dimensions of the hypercube. The single processor execution times were recorded for determining the speedup performance of the other dimensions. Since the total computational load could not be controlled directly in terms of time, iterative recalculation of the matrix was used as the control mechanism. For each size of the hypercube, the program was executed 5 times for varying computational loads. The levels of the computational loads were expressed as the number of times the elements were recalculated. Table 2 gives the levels of control used in the benchmark. The message traffic load was controlled with the number of processors and the number of messages is an aggregate message count. In all, 270 observations of execution time were recorded.

*Data Collection* Each node recorded its computational and message processing times in the system log. The computational time required of a given load for a cube size was taken to be a mean of the individual processors' computational time. This is reasonable since each processor had the same load; the computational times recorded by the processors were all within 5 milliseconds of each other and the resolution of the clock is also 5 milliseconds. The message processing time, however, was taken

Table 2. Benchmark Levels of Control

<i>Computational Load (x)</i>	<i>Number of Nodes</i>	<i>Number of Messages</i>
5	1	0
7	2	2
10	4	12
16	8	56
20	16	240
30	32	992
65		
90		
200		

to be the maximum time of all the processors in the cube. This is also reasonable since the process was not considered complete until the last message was received. Total execution time is the sum of the two times.

#### *Simulation Model*

A simulation model was constructed using SLAM II Simulation Language [PR86]. The model simulates a process running in parallel on a hypercube of a chosen size ranging from 0 to 5. The computational load and message load is balanced across all processors. Each node executes a specified number of bursts and each burst is followed by a message being sent to any number of predetermined or randomly chosen receivers.

*Model Components* To construct the model, times for three components had to be determined. The first component is the computational time per node which was assumed to be about  $1/N$ th of the total computational load. The benchmark

confirmed that this was a true assumption. The second and third components are interprocessor communications time and overhead due to implementation details. Interprocessor communications time is addressed below however; the last component is addressed with model validation since it is related to validation and calibration of the overall simulation model.

Equation 7 found in Chapter II estimates interprocessor communication times for nearest neighbors. Since the simulation model simulates both nearest-neighbor and non-nearest-neighbor communications, the use of Equation 7 in the simulator was insufficient. The iPSC uses a packet switched network with predetermined routing so non-nearest-neighbor communications can be thought of as a series of nearest-neighbor transmissions. Along the sender/receiver path, intermediate nodes perform the store-and-forward function. Equation 7 can be modified to model interprocessor communications over multiple links

$$T_{ML} = t_i + LHt_c + It_i \quad (8)$$

where  $H$  is the number of hops or links traversed,  $I$  is the number of intermediate nodes visited and  $t_i$  is the time required for the node to forward the message.  $H$  and  $I$  are directly related since the number of intermediate nodes visited is one less than the number of hops in the sender/receiver path. When there is a single link in the path, Equation 8 reduces to Equation 7.

To find the actual message transmission times, a benchmark program was executed on the iPSC. One message was passed back and forth between a sender/receiver pair 100 times (200 transmissions). Two hundred transmissions were chosen to overcome the resolution of the clock. The transmissions were done in a sterile environment: first, there were no overlapped transmissions of the message since the both nodes were required to wait on the message before it could be returned to the other, and second, there were no other communications in the cube nor were there any other processes running. Twenty messages of varying sizes (up to 1024 bytes) were passed between 31 sender/receiver pairs, thus the data set consisted of 620 data

## Message Transmission Times

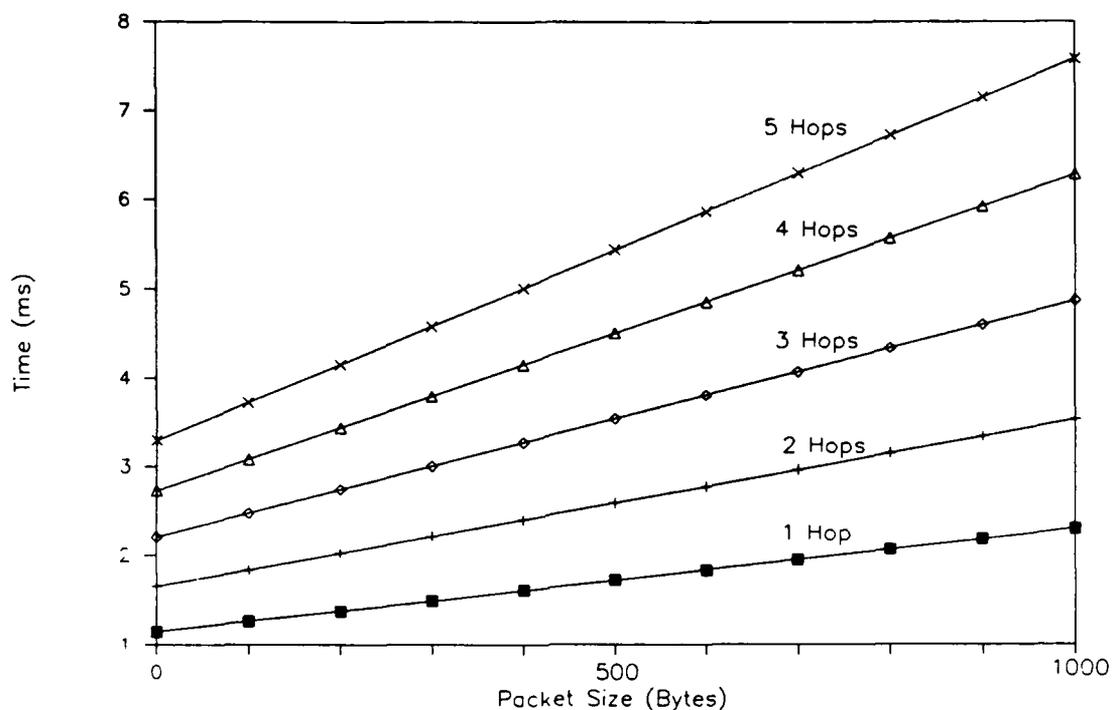


Figure 4. Plot of Message Transmission Times

points. Figure 4 shows the linear relationship between messages requiring the same number of hops. Using SAS, Equation 8 was estimated from the data producing

$$T_{ML} (ms) = 1.1232 + 0.0008968LH + 0.485I \quad (9)$$

The coefficient of determination for this model is .9939.

In comparison to the results obtained by Reed and Grunwald, the message latency is slightly lower than the 1.7 milliseconds they reported. Even more surprising was the difference in transmission time per byte; 0.9 microsecond is significantly faster than the 2.8 microseconds reported. Later research by Reed yielded a new estimation of Equation 7 to be 0.706 millisecond for latency and 0.519 microsecond for transmission time per byte [RE88]. He attributed the difference to a revision in the node operating system which handles the message processing.

*Model Construction* The SLAM II source code modelling parallel processing on a hypercube is listed in Appendix B. The basis for modelling communications in the hypercube was Equation 9 since non-nearest-neighbor communications can be thought of as a series of nearest-neighbor communications along the sender/receiver path. The time spent at each intermediate node is the coefficient of the third term in Equation 9, while the transmission time per link is the size of the message multiplied by the coefficient of the second term.

Each node in the cube was modelled as a unique RESOURCE. The communications links outbound from a node were modelled with a single server ACTIVITY proceeded by a QUEUE. Usually, each outbound link would be modelled with its own server and queue, but the iPSC has a peculiar hardware characteristic. If more than one physical channel transmits at a time, packets are lost; therefore, only one channel is allowed to transmit at a time [IN86b]. For packet transfers the iPSC uses a predetermined routing scheme based on the logical exclusive-or operation of the nodes' identification number. This routing algorithm was easily implemented with a FORTRAN function that is visible to SLAM II. Communications has priority over normal node processing, so packets arriving at a node have the ability to pre-empt a node's processing. The implementation of this condition was trivial in SLAM II by allowing a packet to PREEMPT a task currently utilizing the node-RESOURCE. SLAM took care of queuing the messages that arrive at the nodes for processing.

The concurrent activities of a parallel process on a two-dimensional hypercube are shown in Figure 5. Activities that occur in the unshaded areas are concurrent. The shaded areas represent the activities that required the use of the node; for example, a node could process messages or it could process a task, but it could not do both simultaneously. Figures 6 and 7 show the flow of events in the simulation model which occur as follows:

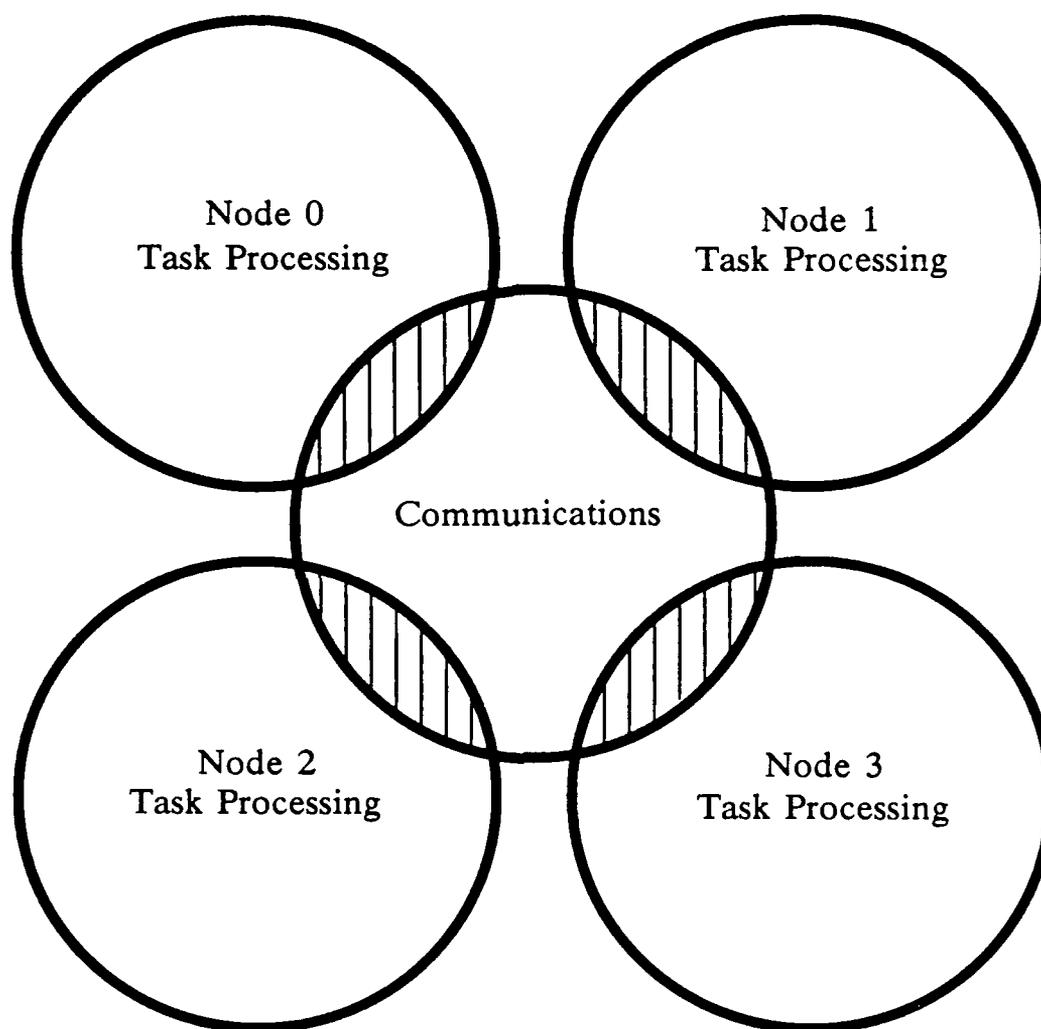


Figure 5. Concurrent Activities

1. A process enters the cube.
2. The process is partitioned into  $N$  tasks;  $N$  can be  $\{1,2,4,8,16,32\}$ .
3. Each task entity is assigned an identification number which is conveniently the identification number of the node to which it is assigned.
4. The start time of the process is recorded.
5. Tasks wait until their node become available.
6. Tasks are processed for some length of time. Burst lengths are independent random variables from the same distribution.
7. Following the burst, a task communicates its results to some specified number of other tasks by replicating itself into a message entity for each message it sends.
8. Once the last message is sent, the task releases the node.
9. If a task is not complete, it waits for node to which it is assigned and it is processed again. When the task is completed, the task entity is terminated.

Meanwhile:

10. A message entity is assigned the next node it must go to enroute to its destination.
11. A message entity enters the transmit queue at its current node.
12. A message is transmitted: from Equation 9, the transmission time is 0.9 microseconds for every byte of information in the message.
13. At the next node, a message pre-empts the node's task processing. If this node is the destination, the node receives the message. If this node is not the destination, intermediate-node processing occurs and the next node in the path is assigned and the message continues to traverse the network.
14. The node is released to return to task processing.

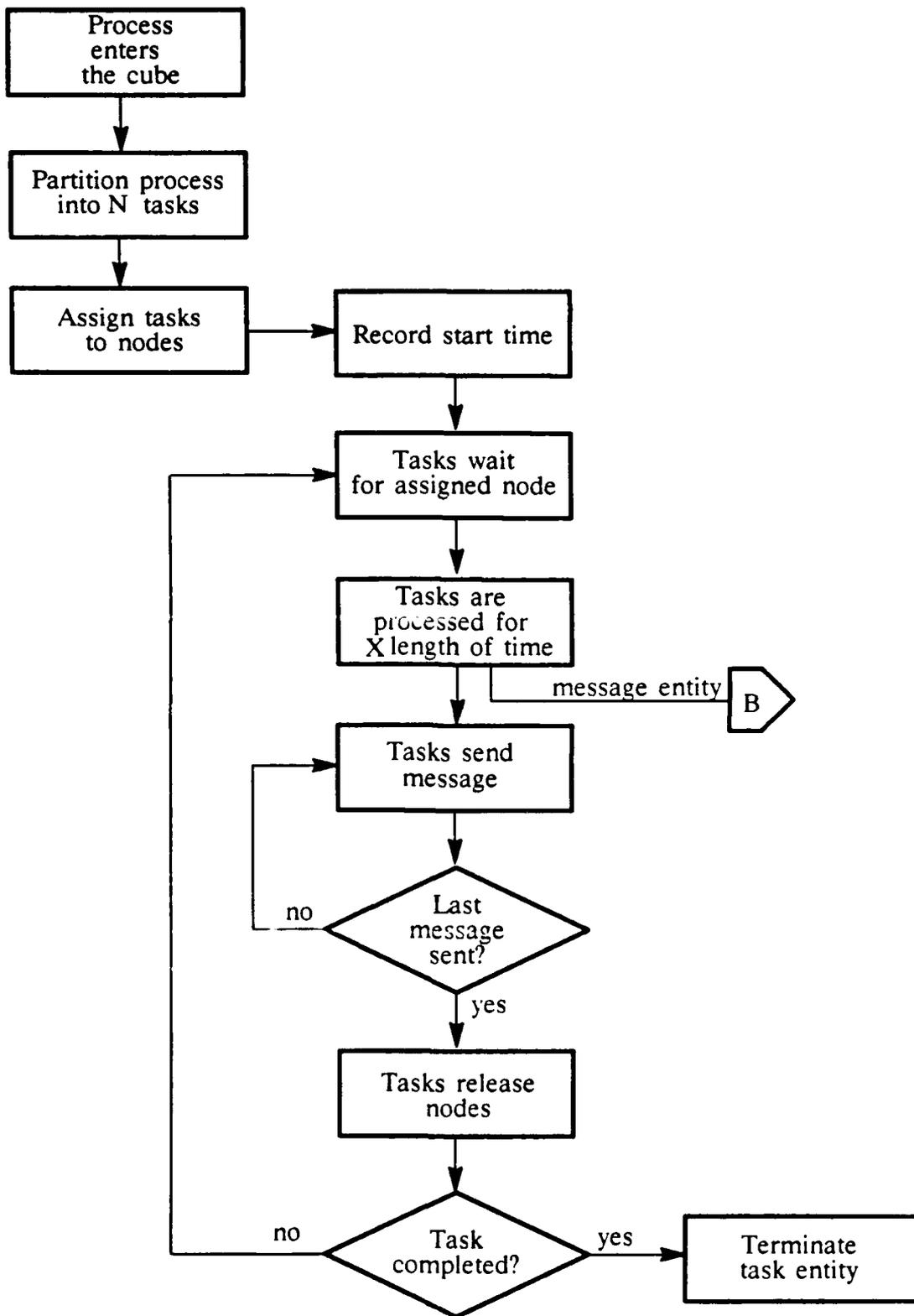


Figure 6. Simulation Flow of Events (Part A)

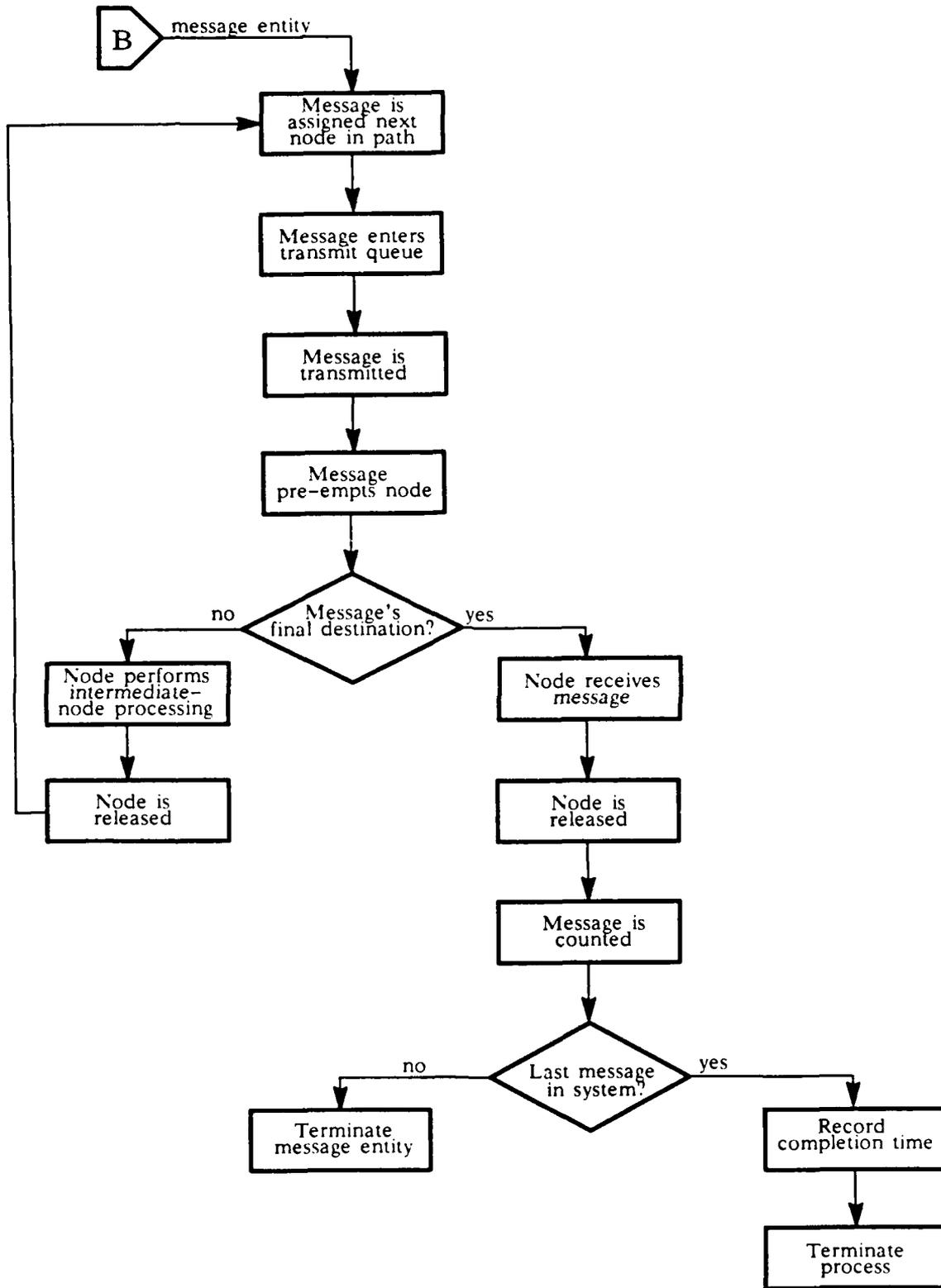


Figure 7. Simulation Flow of Events (Part B)

15. The message is counted and terminated if it is not the last one.
16. When the last message is received, the completion time is recorded and the process is terminated.

*Model Validation* Before the model could be validated with the benchmark, it had to be calibrated to the exact implementation of the benchmark program. There was overhead introduced into the benchmark program from measuring the clock and checking the receipt of messages. Message checking ensured that all messages were received and the data was stored in the appropriate place in the matrix. This additional workload required computation time that must be accounted for in the simulation model.

To find the calibration time, the actual time required to execute message checking was measured in the benchmark program for all dimensions of the hypercube. The time required for the process to actually receive the message was subtracted out, thus leaving the overhead associated with checking flags. Using linear regression, the calibration function for  $N > 2$  was estimated to be

$$C = 5 + 7.5N \quad (10)$$

where  $C$  is the amount of time, in milliseconds, required to perform overhead and  $N$  is the number of nodes in the cube. For the two-node case, the overhead was negligible.

The calibration function was incorporated into the model for validation purposes. The simulation was exercised with four uniprocessor loads which were taken from actual uniprocessor times observed in the benchmark experiment. The individual node processing times were taken from a normal distribution with  $\mu$  computed as the uniprocessor load divided by the number of nodes and  $\sigma$  equal to 2.5 milliseconds since the benchmark showed that the computational times of the individual nodes were all within 5 milliseconds of each other. The total processing time for each node was completed in one burst which simulates the benchmark program. For

Table 3. Uniprocessor Loads Used for Validation

<i>Uniprocessor Load</i>		
<i>Raw Time</i>	<i>Recalculations</i>	<i>Number of Nodes</i>
200	5	2
1205	30	4
3615	65	8
8010	200	16
		32

each uniprocessor load, five runs were made for cube sizes of 1 through 5, for a total of 100 simulation times. Table 3 shows how the 100 simulation times were collected. The uniprocessor loads are also expressed in terms of the number of recalculations in the benchmark program.

The five runs for a given uniprocessor load and cube size were averaged as well as the five runs of the benchmark program corresponding to the uniprocessor load and cube size. The first research hypothesis listed in Table 1 was tested with a paired difference  $t$  test at a confidence level of 0.1. The raw times were used to perform the test since the speedup within each pair is relative to the same the uniprocessor time.

#### *Experiment Design*

To test the second and third research hypotheses listed in Table 1, an experiment was designed which exercised the simulation model without the calibration function. The uniprocessor load was varied across two levels. The message traffic load was varied by introducing two more variables that quantify the amount of message traffic. These two variables were the number of bursts and the number of

Table 4. Control Variables

<i>Uniprocessor Time</i>	<i>Nodes</i>	<i>Bursts</i>	<i>Receivers</i>
1000	2	1	$N / 2$
20000	4	5	$N - 1$
	8		
	16		
	32		

messages sent per burst which was expressed as a function of the number of nodes. Two levels of each were selected.

The general linear model for total execution time is

$$T_{CL} = \mu + \frac{U}{N} + NBR + \text{error} \quad (11)$$

where  $\mu$  is the experimental average,  $U$  is the uniprocessor time,  $N$  is the number of nodes,  $B$  is the number of bursts and  $R$  is the number of receivers. The general linear model for speedup is

$$\log S_{CL} = \log U + \log T_{CL} + \text{error} \quad (12)$$

*Levels of the Control Variables* The control variables were set to levels shown in Table 4. Each experimental unit consisted of a uniprocessor time, a cube size, a level of bursts and a level of receivers. In the case where  $N = 2$ , both levels of  $R$  were the same. Excluding the duplicated units, there was a total of 36 experimental units. The computational time for each node was randomly selected from a normal distribution. In the case of  $R = N - 1$ , every node sent to every other node, but when  $R = N/2$ , the receiving nodes were randomly selected. When the number of bursts was 1, the burst length was the node's computational time, but when the

number of bursts was 5, the burst duration was randomly selected from a normal distribution.

*Data Collection* For each experimental unit, 10 runs were made. To test the third research hypothesis, ten additional runs were made for the experimental units that include 5 bursts, but the burst lengths were randomly chosen from an exponential distribution. Table 5 summarizes the entire experiment design.

Table 5. Experiment Design

<i>Runs</i>	<i>Burst Length</i>	<i>Uniprocessor Time</i>	<i>Number of Nodes</i>	<i>Number of Bursts</i>	<i>Receivers</i>
10	Normally Distributed	1000 20000	2 5	1	$N - 1$
10	Normally Distributed	1000 20000	4 8 16 32	1 5	$N / 2$ $N - 1$
10	Exponentially Distributed	1000 20000	2	5	$N - 1$
10	Exponentially Distributed	1000 20000	1 8 16 32	5	$N / 2$ $N - 1$

### *Summary*

This chapter explained the procedure followed in this research. A hypercube was benchmarked using a matrix multiplication algorithm. A discrete event simulation of a parallel process running on a hypercube was then constructed and validated with the benchmark data. Finally, an experiment was designed that exercised the simulation model so that the functional relationship between workload characteristics and speedup characteristics could be determined.

## *IV. Results*

This chapter presents the results of the benchmark and the discrete event simulation experiment and evaluates the research hypotheses listed in Table 1.

### *Evaluation of the First Research Hypothesis*

Results of the benchmark experiment are given in Appendix C. Table 6 shows the average times for selected computational loads which are expressed in terms of the number of recalculations. The results of the simulation model with the calibration function incorporated are listed in Appendix D. Table 6 also shows the average times obtained from the simulation. Figure 8 shows the standardized deviations of the simulation times  $t$  on the benchmark times for the uniprocessor loads listed in Table 6. The test statistic for the paired difference  $t$  test was 0.22, therefore, the first research hypothesis was not rejected and it was concluded that the simulation model was an accurate representation of hypercube performance for the matrix multiplication workload.

Table 6. Benchmark Versus Simulation Times

<i>Recalculations</i>	<i>Uniprocessor Time</i>	<i>Nodes</i>	<i>Total Time (ms)</i>	
			<i>Benchmark</i>	<i>Simulation</i>
5	200	2	104	104
		4	104	95
		8	130	112
		16	169	184
		32	318	351
30	120	2	606	607
		4	349	346
		8	255	238
		16	224	247
		32	398	382
65	3615	2	1812	1814
		4	944	950
		8	533	540
		16	391	398
		32	445	458
200	8010	2	4009	4009
		4	2056	2048
		8	1095	1089
		16	665	672
		32	591	595

## Benchmark Versus Simulation

Standardized Deviation

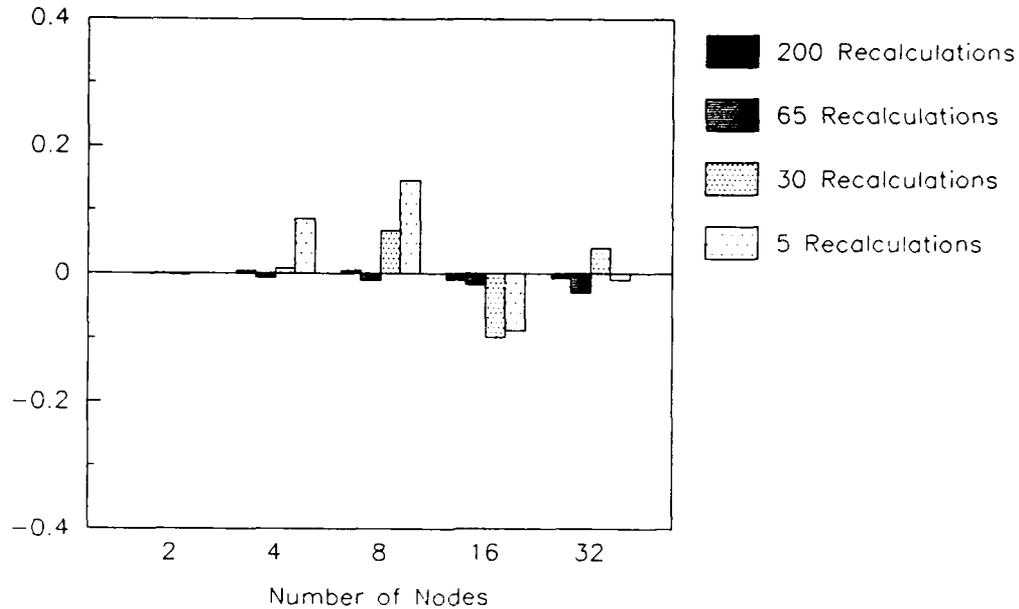


Figure 8. Comparison of Benchmark and Simulation Times

*Benchmark Results* From the benchmark data contained in Appendix C, two equations were estimated. Figure 9 shows the mean computational time, communications time and total execution time for 16 recalculations of the matrix. From 2 to 32 nodes, the message processing time increased almost linearly with the number of nodes while the computation time decreased inversely with the number of nodes. This graph suggests a linear communication cost model similar to Stone's [ST87]. The equation for the total execution time was estimated to be

$$T_b = -4.7 + 1 \left( \frac{U}{N} \right) + 10.64N \quad (13)$$

The coefficient of determination for this model is .9998. The graph also shows that the message processing time for a single node is zero and negligible for two nodes.

# Hypercube Benchmark

Matrix Recalculated 16 Times

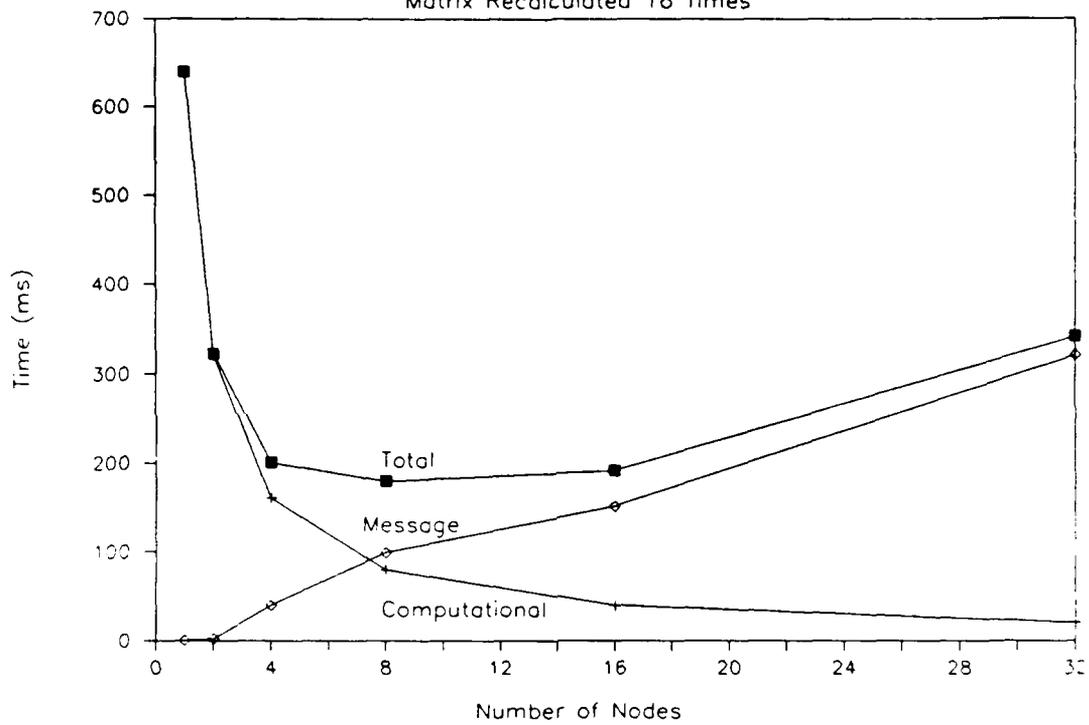


Figure 9. Plot of Total, Computational and Message Processing Times

This accounts for the negative constant in the model which is required to fit the rest of the data.

This model is similar to Stone's since the uniprocessor time is, in terms of his variables,  $RM$ . Equation 13 can also be minimized with respect to  $N$  by setting the first derivative equal to zero

$$\frac{dT_b}{dN} = -\frac{U}{N^2} + 10.64 = 0 \quad (14)$$

and solving for  $N$

$$N = \sqrt{\frac{U}{10.64}} \quad (15)$$

Although derived in a different manner, the relationship between the optimal level of  $N$  and both the uniprocessor time and the communications time is the same relationship that was given by Stone in Equation 6. The communications cost of 10.64 milliseconds per node included the actual communication times and the overhead associated with checking the receipt of messages as discussed earlier.

Although Equation 13 accurately models the total execution time in terms of the uniprocessor load and the number of nodes, information about the message traffic load is embedded in the number of nodes and the assumption that every node sends once to every other node. Equation 11 allows the assumption about the message traffic load to be relaxed since nodes can send one or more messages to a subset of the cube. Equation 11 was estimated to be

$$T_{CL} = 46.7 + 1 \left( \frac{U}{N} \right) + .03 NBR \quad (16)$$

For the benchmark program,  $B = 1$  and  $R = N - 1$ . In this model, the three primary independent variables which describe the workload were used. The coefficient of determination for this model is .9992. Figure 10 compares the predictive power of the two models. Both models indicate that the computational time decreases with the number of nodes, but communication time increases as more nodes are added. Figure 11 shows the speedup curves for four different uniprocessor loads.

## Hypercube Benchmark

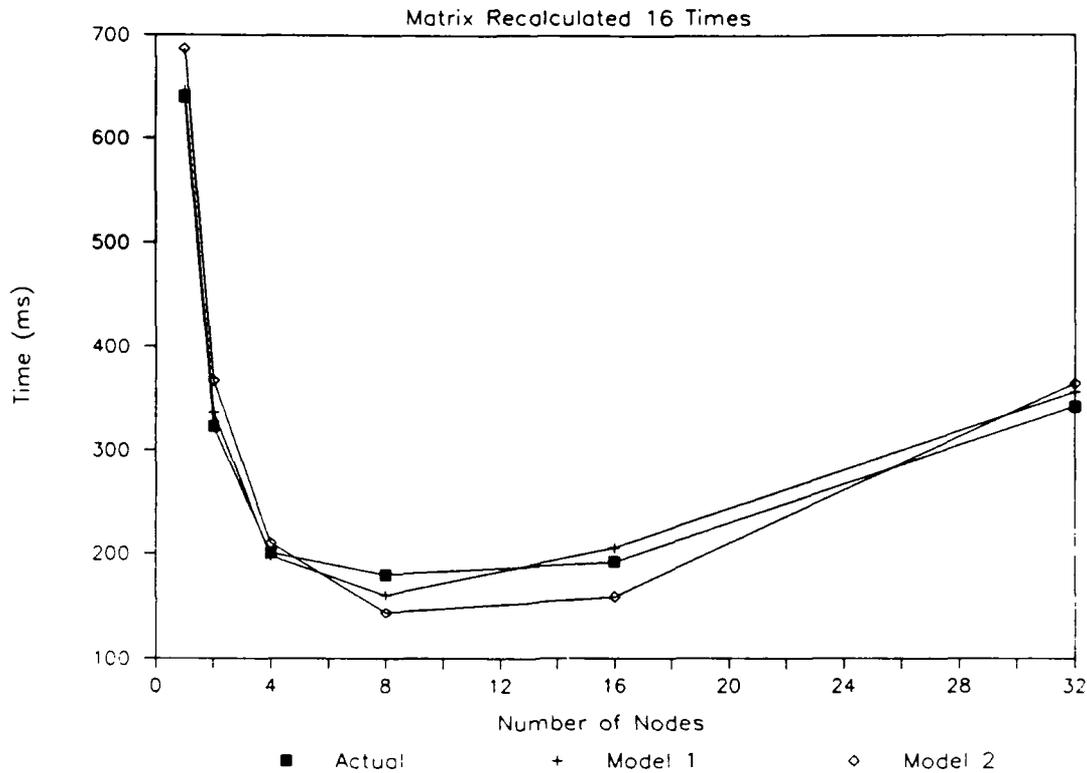


Figure 10. Comparison of Models

The point where each curve turns down indicates which level of  $N$  minimizes Equations 13 and 16 relative to the uniprocessor load.

*Computation to Communications Ratio* The benchmark data can be used to derive another computational to communications ratio. The speedup achieved in the benchmark in terms of  $N$  was

$$S_b = \frac{U}{\frac{U}{N} + 10.64N - 4.7} \quad (17)$$

If  $S_b$  is set equal to 80% of the ideal speedup, then solving for  $U$  yields

$$U = 42.4 N^2 - 18.8 \quad (18)$$

### Speedup of Four Computational Loads

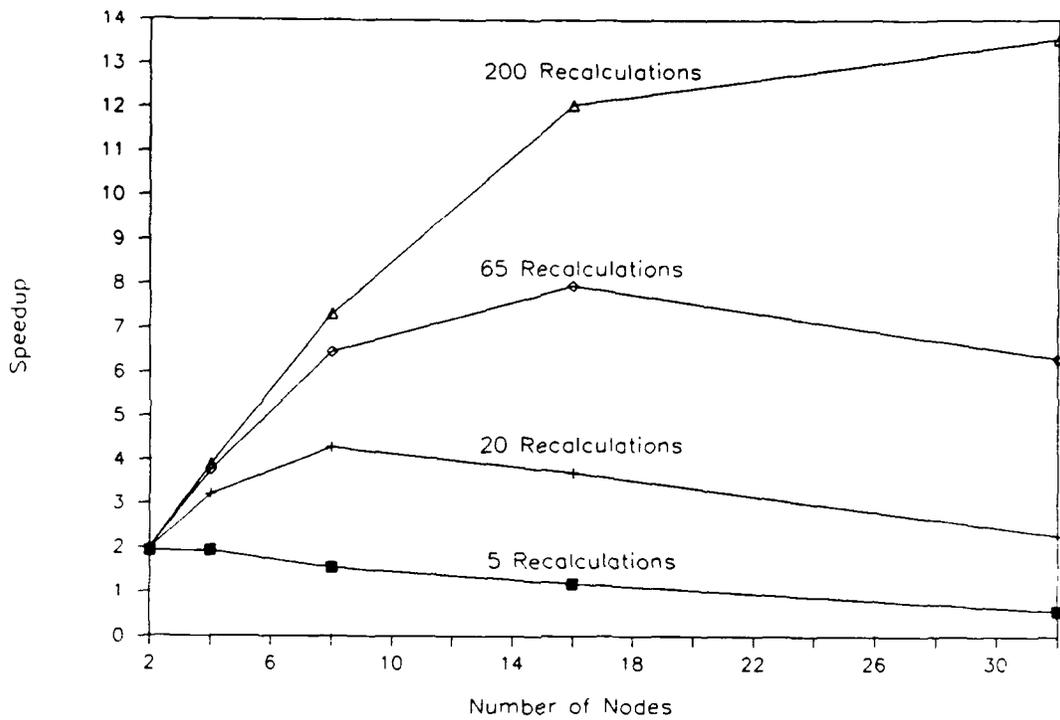


Figure 11. Plot of Speedup for Four Computational Loads

Therefore, to achieve 80% of ideal speedup, the ratio of computational time per node to total communications time is approximately 4 to 1 since

$$\frac{42.4 N^2 - 18.8}{10.64 N^2} \approx 4$$

To reach 80% of the ideal speedup in the case of a balanced load over 4 or more processors, the computational time per node must be 4 times as great as the total communications time. Said differently, the uniprocessor time to total communications time must be  $4N$  to 1. For example, if each node sends one message to every other node after completing its computation and 4 nodes were used, then the uniprocessor time must be about 660 milliseconds to obtain a speedup of 3.2; however, for 8 nodes, the uniprocessor time must be about 2700 milliseconds to achieve a speedup of 6.4. As more communication time is required, either by nodes sending more messages or more nodes being added to the system, the uniprocessor time must also increase to maintain the same speedup. When  $N = 2$ , the communications time is not significant; so near ideal speedup will be achieved.

#### *Evaluation of the Second Research Hypothesis*

Since the first hypothesis was not rejected, the data obtained from the simulation model was analyzed with confidence that it was representative of the hypercube architecture. The actual times obtained from the simulation are optimistic since the model does not account for processing time required for a task to verify receipt of messages. The simulation can, however, show relative speedup for varying workloads.

The average execution times obtained from the simulation when the burst lengths are normally distributed are given in Table 7. Appendix E contains the complete data set. Figures 12 and 13 compare the speedup achieved for each level of  $B$  and  $R$  for the 1000-millisecond and 20,000-millisecond loads respectively.

Table 7. Simulation Results

<i>Number of Bursts</i>	<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Time (ms)</i>	
			<i>U = 1000</i>	<i>U = 20,000</i>
1	4	2	273	5325
	8	4	151	2757
	16	8	95	1414
	32	16	91	760
	2	1	517	10270
	4	3	275	5327
	8	7	157	2735
	16	15	113	1434
	32	31	135	805
	5	4	2	291
8		4	192	2797
16		8	188	1506
32		16	292	960
2		1	526	10280
4		3	301	5357
8		7	231	2839
16		15	283	1604
32		31	513	1183

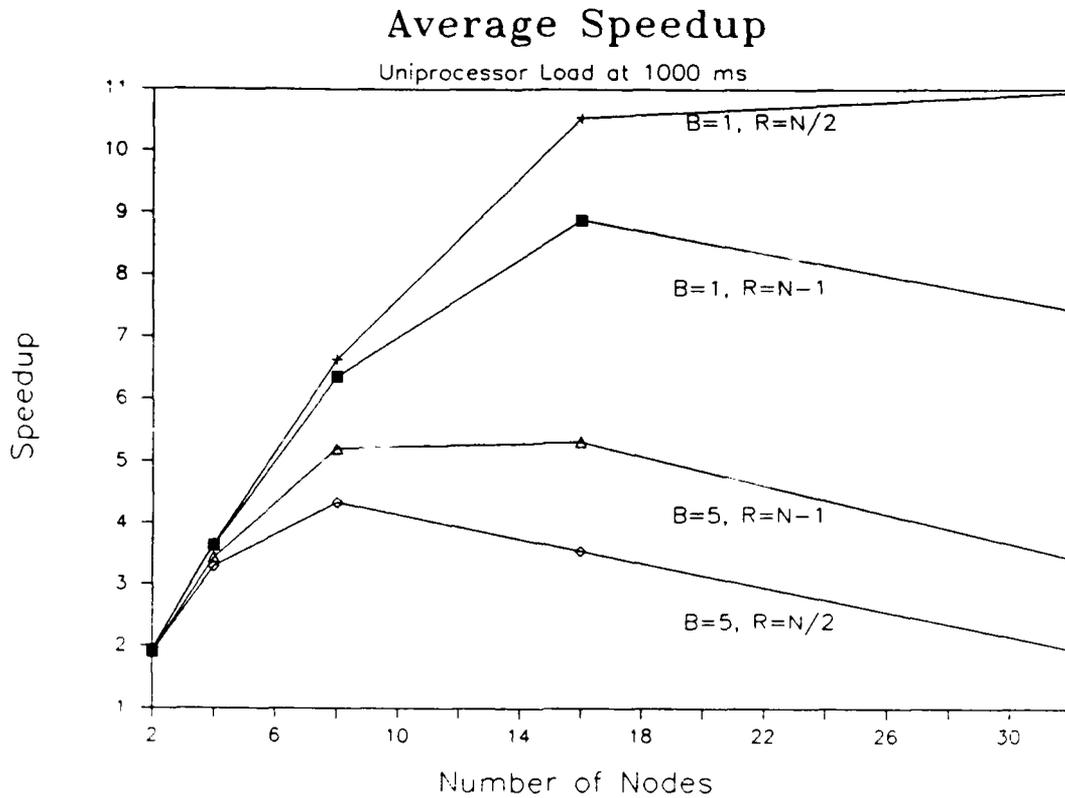


Figure 12. Plot of Speedup for Total Computational Load of 1000 ms

Equation 11 was re-estimated with the simulation data and yielded the following results

$$T_s = 77 + 1 \left( \frac{U}{N} \right) + .088NBR \quad (19)$$

The coefficient of determination for this model is .9955 and all terms are significant at a confidence level of .01. Figure 14 compares the actual and predicted execution times when the uniprocessor load was 1000 milliseconds. The difference between this equation and Equation 16 is attributed to the absence of message checking in the simulation. This relationship implies that as nodes are added to the process, the execution time will decrease, but the decrease is offset by an increase in the message traffic load. Since the total execution time can be transformed into speedup

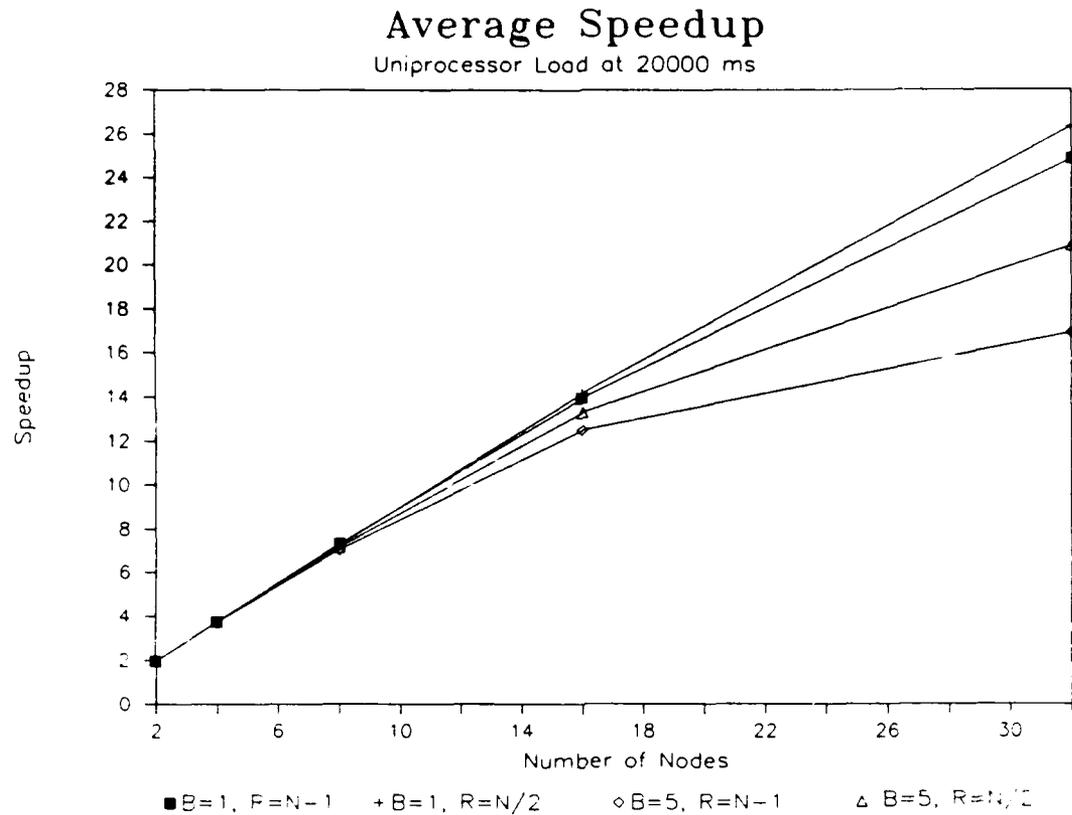


Figure 13. Plot of Speedup for Total Computational Load of 20,000 ms

and regression analysis revealed a functional relationship between the independent variables, the second research hypothesis was rejected. Equation 19 indicates that the total computational load, number of nodes and message traffic load all affect the speedup of a process run in parallel on the hypercube.

## Actual and Predicted Values

Uniprocessor Load at 1000 ms

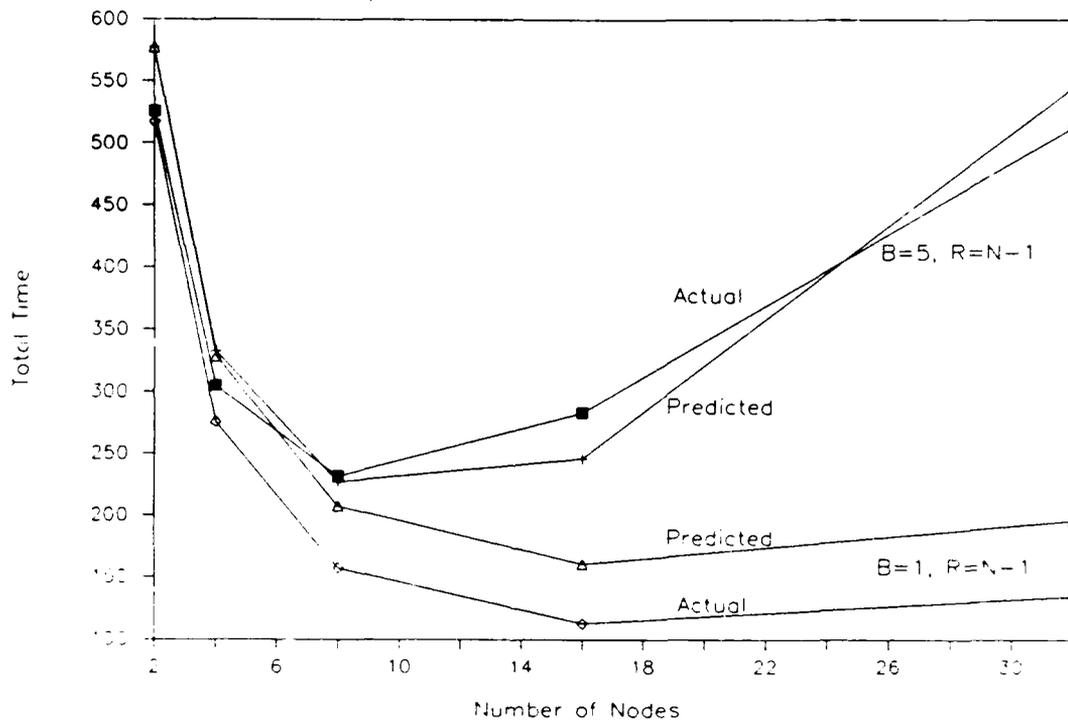


Figure 14. Comparison of Actual and Predicted Values

### *Evaluation of the Third Research Hypothesis*

The total execution times collected from the simulation when the burst lengths were exponentially distributed are given in Appendix E. For every experimental unit at  $R = N - 1$ , the results were identical to the normally distributed burst lengths. When  $R = N/2$ , there were very slight differences in the total execution time. The raw data indicated that there was no difference in the total execution time of a process explained by the burst times, so the third hypothesis was not rejected.

## V. Conclusions

### *General*

The communication costs incurred by parallelization reduces the speedup of a process run on a hypercube. Although this is not a novel idea, this thesis presented the functional relationship between the workload characteristics that affect the speedup of a process on a hypercube architecture. The methodology presented here is directly applicable to any architecture.

### *Research Results*

The speedups achieved in both the benchmark and the calibrated simulation were not significantly different. The data listed in Table 6 supported the first research hypothesis. Since the simulation model could be validated with the benchmark program, it was assumed to be an accurate representation of hypercube performance.

The speedups obtained from the simulation changed with the workloads placed on the cube. The functional relationship between the workload characteristics and the total execution time was given in Equation 19; therefore, the second hypothesis was rejected. When the experiment was repeated with the exponentially distributed burst lengths, the data was not significantly different, so the third hypothesis was not rejected.

### *Benchmarking Versus Simulation*

The benchmark of an actual hypercube provided some insight into the speedup phenomenon. Creating controllable workloads is cumbersome, so a simulation model of hypercube processing was constructed, verified and validated. Unlike benchmarking, the simulation allowed more flexibility and control of the workload characteristics. The model did not account for overhead processing innate to an actual

implementation of a parallelized algorithm. Although benchmarking captures the overhead processing, it will change from algorithm to algorithm and with the chosen implementation. As shown by the benchmark, the overhead processing cannot be ignored since it also affects speedup. Simulation, on the other hand, provided a means for studying the performance of the architecture in terms of workload characteristics by removing the implementation and algorithm variable. Simulation is useful for comparing potential algorithms implemented on the hypercube. This requires the algorithm to be described only in terms of the workload it places on the nodes and the communications network.

#### *Impact of Workload on Speedup*

The results obtained from the simulation were not surprising. Clearly, if the computational workload per node is large compared to the communication time required, then speedups will be near ideal as shown in Figure 13. At 32 nodes, the speedup was about 28 when the message load was at the lowest level, but the speedup dropped to approximately 16 when the message load was at the highest level. At 16 nodes, there was little difference in speedup since the total message traffic load at all four levels was small compared to the computational load on each node. When the total computational load was 1000 milliseconds, the impact of message traffic load was felt at 8 nodes, as shown in Figure 12. All the data collected, either from the simulation or from the benchmark, supported a computational time per node to total communications time ratio of 4 to 1.

The results obtained when the burst lengths were exponentially distributed suggest that as long as the total computational workload and the message traffic load are balanced, the points in time when computation and message processing occur have no affect on the speedup. This means that the amount of workload placed on each processor is the same, but the behavior of each workload can be very different. Assuming that process synchronization is not an issue, this finding implies

that the goal of a decomposition strategy should be to balance the load without regard to homogeneous behavior. Process synchronization was not modelled, so balancing some workloads may impose additional time for synchronization.

#### *Suggestions for Further Research*

This thesis has shown the effect of the message traffic load on speedup. Previous research has shown that speedup is degraded by an unbalanced computational workload. Another issue to study would be the effects of an unbalanced message traffic load and a balanced computational load on the speedup of a process run in parallel. Perhaps slight imbalances in computational loads could be offset with complementary imbalances of message loads. The message traffic generated when the burst lengths were normally distributed was not at a high enough intensity to saturate the network. How would speedup be affected if the assumption about single packet messages were loosened to allow multiple packets so that the message traffic load could saturate the network; and, could network saturation be overcome by exponentially distributed burst lengths? The use of simulation opens a door for studying the impact various process behaviors have on speedup for any architecture.

#### *Summary*

This thesis has successfully demonstrated the use of benchmarking and simulation to determine the effects of the workload on the speedup of a process run in parallel on the hypercube architecture. Additionally, simulation was shown to be a viable tool for investigating the speedup phenomenon by providing flexibility and easy control of the independent variables.

## Appendix A. *Benchmark Program*

This appendix contains the listing of the node program used to benchmark the iPSC. The second file included, "declare.h" contains declarations used by both the nodes and the host, and is listed after the program. A copy of this program is downloaded by the host to each node in the cube.

```
#include "/usr/ipsc/lib/cnode.def"
#include "declare.h"

/* global variables */
int  my_pid,my_node; /* local process and node number */
int  nprocs;        /* number of processors in cube */
int  host_chan     /* channel for host-node communication */
int  node_chan     /* channel for node-node communication */
int  cnt;          /* incoming message size */
int  fr_node;     /* node message came from */
int  fr_pid;      /* process ID that message came from */
int  msg_length;  /* outgoing message size */
long clock();    /* clock reading */
long start_time; /* starting time */
long stop_time;  /* time process finishes completely */
long end_mult;   /* time process finishes computing matrix */
char msgbuf[80]; /* buffer for messages to system log file */
float result[SIZE][SIZE]; /* resulting matrix */
```

```

main ()
{
    setup ();          /* open communication channels */
    multiply();        /* compute matrix and pass results */
    sendw (host_chan, UPLOAD, result,
           SIZE * SIZE * sizeof (float), HOST,
           HOST_PID); /* send results to host */
    cclose (host_chan);
    sprintf (msgbuf, " %ld %ld", stop_time - start_time,
            end_mult - start_time);
    syslog (my_pid, msgbuf); /* total time, cpu time */
}

```

```

/* Open communication channels and receive size of cube */
/* from the host. */

```

```

setup()
{
    my_pid = mypid ();
    my_node = mynode ();
    host_chan = copen (my_pid);
    node_chan = copen (my_pid);

    recvw (host_chan, PARAM, &nprocs, PARAM_MSG_SIZE,
           &cnt, &fr_node, &fr_pid);
    msg_length = sizeof(float) * (SIZE*SIZE) / nprocs;
}

```

```

/* Compute the matrix. */

multiply()
{
    int  msg_rec[32]; /* flag vector for message status */
    int  msg_type;    /* next two variables are used to */
    int  ans;         /* check the status of messages */
    int  count;       /* number of messages expected */
    int  node;        /* destination of a message */
    int  i, j, k;     /* loop counters and matrix indices */
    int  recomp;      /* loop counter for recalculations */
    int  f_row;       /* index of first row of submatrix */
    int  l_row;       /* index of last row of submatrix */
    int  cols;        /* number of columns in submatrix */
    int  f_col;       /* index of first column */
    int  l_col;       /* index of last column */
    float *bufptr;    /* place to put incoming results */
    float temp;       /* temporary storage used for */
                    /* recalculating matrix */

    /* Compute the indices of the the submatrix based on the */
    /* number of processors and nodes' identification number. */

    cols = SIZE * SIZE / nprocs;
    f_row = my_node * SIZE / nprocs;
    l_row = ((my_node + 1) * SIZE - 1) / nprocs;

```

```

f_col = (my_node * cols) % SIZE;
l_col = ((my_node + 1) * cols - 1) % SIZE;
bufptr = &result[f_row][f_col];
temp = 0.0;

start_time = clock();
for (recomp = 0; recomp < CPU_LOAD; recomp ++)
    for (i = f_row; i <= l_row; i ++)
        for (j = f_col; j <= l_col; j ++)
            {for (k = 0; k < SIZE; k ++)
                temp = temp + MATRIX [k][j] * MATRIX [i][k];
                result [i][j] = temp;
                temp = 0.0;
            }
end_mult = clock();

for (node = 0; node < nprocs; node ++)
    if (node != my_node)
        send (node_chan, my_node, bufptr, msg_length,
            node, NODE_PID);

for (i=0; i < nprocs; i++)
    msg_rec[i] = 1;

count = nprocs - 1;
msg_type = 0;
do
    {if (msg_rec[msg_type] && (msg_type != my_node))

```

```

{ans = probe (node_chan, msg_type);

if (ans >= 0)
    {bufptr = &result[msg_type*SIZE/nprocs]
      [(msg_type * cols) % SIZE];
      recv (node_chan, msg_type, bufptr,
            msg_length, &cnt, &fr_node, &fr_pid);

      count --;
      msg_rec[msg_type] = 0;
    }
}
msg_type = (msg_type + 1) % nprocs;
}
while (count);
}

```

```

/* message types */
#define PARAM 40 /* host-to-node: cube dimension */
#define UPLOAD 70 /* node-to-host: upload results */
#define PARAM_MSG_SIZE sizeof (int)

/* cube definitions */
#define HOST_PID 1
#define NODE_PID 1
#define ALL_NODES -1
#define HOST 0x8000

/* parameter definitions */
#define SIZE 8 /* size of square matrix */
#define CPU_LOAD 5 /* number of recalculations */

#define nextarg { argc--; argv++; } /* used in host program */

float MATRIX [SIZE][SIZE] =
{ {0.2, 0.3, 0.1, 0.4, 0.5, 0.7, 0.9, 1.0},
  {0.1, 0.2, 0.6, 0.1, 0.2, 0.1, 1.0, 0.5},
  {0.5, 0.2, 0.1, 0.2, 0.5, 0.7, 0.1, 0.9},
  {0.2, 0.3, 0.2, 0.3, 0.8, 0.4, 1.0, 0.2},
  {0.8, 0.6, 0.4, 0.2, 0.5, 0.7, 0.2, 0.1},
  {0.5, 0.2, 0.6, 0.5, 0.9, 0.6, 0.4, 0.6},
  {1.0, 0.1, 0.1, 0.7, 0.8, 0.5, 0.8, 0.9},
  {1.0, 0.3, 0.6, 0.9, 1.2, 0.3, 0.2, 0.1}
};

```

## Appendix B. *SLAM II Source Code*

This appendix contains the SLAM II source code and the FORTRAN function used for message routing. This source listing is set up to simulate 32 nodes processing 5 burst. The burst lengths are normally distributed and the number of receivers is 31 ( $N - 1$ ). The uniprocessor load is 1000 milliseconds.

```
GEN,CATHY,HYPERCUBE SIMULATION,3/20/88,10,N,N,,,Y/1,72;
LIMITS,128,20,5000;
EQUILVALENCE/ATRIB(2),SOURCE/
      ATRIB(3),DESTINATION/
      ATRIB(4),NEXT_NODE/
      ATRIB(5),CURRENT_NODE/
      ATRIB(6),AWA_FILE/
      ATRIB(7),PRE_FILE/
      ATRIB(8),RECEIVER/
      ATRIB(9),CHANNEL/
      ATRIB(10),COUNT;
EQUILVALENCE/ATRIB(11),XMT_QUE/
      ATRIB(12),XMT_ACT/
      ATRIB(13),BURST_TIME/
      ATRIB(14),BURSTS_RMNG/
      ATRIB(15),TIME_RMNG/
      ATRIB(16),MEAN_BRST_LEN/
      ATRIB(17),STD_DEV/
      ATRIB(18),LAST_MSG;
EQUILVALENCE/XX(41),NBURSTS/
```

```

XX(42),NPROCS/
XX(43),XMT_TIME/
XX(44),OVERHEAD/
XX(45),MSG_SIZE/
XX(46),INT_TIME/
XX(47),MSGS/
XX(48),CMP_TIME/
XX(49),MSG_INIT/
XX(50),NFINISHED/
XX(51),MSG_COUNT;

PRIORITY/1,LVF(9)/2,LVF(9)/3,LVF(9)/4,LVF(9)/
5,LVF(9)/6,LVF(9)/7,LVF(9)/8,LVF(9);
PRIORITY/9,LVF(9)/10,LVF(9)/11,LVF(9)/12,LVF(9)/
13,LVF(9)/14,LVF(9)/15,LVF(9)/16,LVF(9);
PRIORITY/17,LVF(9)/18,LVF(9)/19,LVF(9)/20,LVF(9)/
21,LVF(9)/22,LVF(9)/23,LVF(9)/24,LVF(9);
PRIORITY/25,LVF(9)/26,LVF(9)/27,LVF(9)/28,LVF(9)/
29,LVF(9)/30,LVF(9)/31,LVF(9)/32,LVF(9);
ARRAY(1,32)/0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;

;
NETWORK;
RESOURCE/1,NODE_0(1),33,65;
RESOURCE/2,NODE_1(1),34,66;
RESOURCE/3,NODE_2(1),35,67;
RESOURCE/4,NODE_3(1),36,68;
RESOURCE/5,NODE_4(1),37,69;
RESOURCE/6,NODE_5(1),38,70;

```

```
RESOURCE/7,NODE_6(1),39,71;  
RESOURCE/8,NODE_7(1),40,72;  
RESOURCE/9,NODE_8(1),41,73;  
RESOURCE/10,NODE_9(1),42,74;  
RESOURCE/11,NODE_10(1),43,75;  
RESOURCE/12,NODE_11(1),44,76;  
RESOURCE/13,NODE_12(1),45,77;  
RESOURCE/14,NODE_13(1),46,78;  
RESOURCE/15,NODE_14(1),47,79;  
RESOURCE/16,NODE_15(1),48,80;  
RESOURCE/17,NODE_16(1),49,81;  
RESOURCE/18,NODE_17(1),50,82;  
RESOURCE/19,NODE_18(1),51,83;  
RESOURCE/20,NODE_19(1),52,84;  
RESOURCE/21,NODE_20(1),53,85;  
RESOURCE/22,NODE_21(1),54,86;  
RESOURCE/23,NODE_22(1),55,87;  
RESOURCE/24,NODE_23(1),56,88;  
RESOURCE/25,NODE_24(1),57,89;  
RESOURCE/26,NODE_25(1),58,90;  
RESOURCE/27,NODE_26(1),59,91;  
RESOURCE/28,NODE_27(1),60,92;  
RESOURCE/29,NODE_28(1),61,93;  
RESOURCE/30,NODE_29(1),62,94;  
RESOURCE/31,NODE_30(1),63,95;  
RESOURCE/32,NODE_31(1),64,96;
```

;

```
CREATE,,,1;
```

```

ASSIGN,NBURSTS=5,
      NPROCS=32,
      MSGS=NPROCS-1,
      MSG_SIZE=1024,
      CMP_TIME=1000/NPROCS,
      XMT_TIME=0.0008968224,
      INT_TIME=0.4849987,
      OVERHEAD=0.6158445;
ASSIGN,NFINISHED=0,
      MSG_INIT=1.667,
      MSG_COUNT=MSGS*NBURSTS,
      BURSTS_RMNG=NBURSTS,
      XX(70)=NPROCS+1;
;
; PARTITION PROCESS OVER NODES
;
CONT GOON,6;
      ACT,,NPROCS.GE.1,DIM0;
      ACT,,NPROCS.GE.2,DIM1;
      ACT,,NPROCS.GE.4,DIM2;
      ACT,,NPROCS.GE.8,DIM3;
      ACT,,NPROCS.GE.16,DIM4;
      ACT,,NPROCS.GE.32,DIM5;
DIM0 GOON;
      ACT,,,NO;NODE_0
DIM1 GOON;
      ACT,,,N1;NODE_1
DIM2 GOON;

```

```
ACT,,,N2;NODE_2
ACT,,,N3;NODE_3
DIM3 GOON;
ACT,,,N4;NODE_4
ACT,,,N5;NODE_5
ACT,,,N6;NODE_6
ACT,,,N7;NODE_7
DIM4 GOON;
ACT,,,N8;NODE_8
ACT,,,N9;NODE_9
ACT,,,N10;NODE_10
ACT,,,N11;NODE_11
ACT,,,N12;NODE_12
ACT,,,N13;NODE_13
ACT,,,N14;NODE_14
ACT,,,N15;NODE_15
DIM5 GOON;
ACT,,,N16;NODE_16
ACT,,,N17;NODE_17
ACT,,,N18;NODE_18
ACT,,,N19;NODE_19
ACT,,,N20;NODE_20
ACT,,,N21;NODE_21
ACT,,,N22;NODE_22
ACT,,,N23;NODE_23
ACT,,,N24;NODE_24
ACT,,,N25;NODE_25
ACT,,,N26;NODE_26
```

```
      ACT,,,N27;NODE_27
      ACT,,,N28;NODE_28
      ACT,,,N29;NODE_29
      ACT,,,N30;NODE_30
      ACT,,,N31;NODE_31
;
;
NO  ASSIGN,SOURCE=1,
      CURRENT_NODE=1;
      ACT,,,COMP;
;
N1  ASSIGN,SOURCE=2,
      CURRENT_NODE=2;
      ACT,,,COMP;
;
N2  ASSIGN,SOURCE=3,
      CURRENT_NODE=3;
      ACT,,,COMP;
;
N3  ASSIGN,SOURCE=4,
      CURRENT_NODE=4;
      ACT,,,COMP;
;
N4  ASSIGN,SOURCE=5,
      CURRENT_NODE=5;
      ACT,,,COMP;
;
N5  ASSIGN,SOURCE=6,
```

```
                CURRENT_NODE=6;
    ACT,,,COMP;
;
N6    ASSIGN,SOURCE=7,
        CURRENT_NODE=7;
    ACT,,,COMP;
;
N7    ASSIGN,SOURCE=8,
        CURRENT_NODE=8;
    ACT,,,COMP;
;
N8    ASSIGN,SOURCE=9,
        CURRENT_NODE=9;
    ACT,,,COMP;
;
N9    ASSIGN,SOURCE=10,
        CURRENT_NODE=10;
    ACT,,,COMP;
;
N10   ASSIGN,SOURCE=11,
        CURRENT_NODE=11;
    ACT,,,COMP;
;
N11   ASSIGN,SOURCE=12,
        CURRENT_NODE=12;
    ACT,,,COMP;
;
N12   ASSIGN,SOURCE=13,
```

```
                CURRENT_NODE=13;
ACT,,,COMP;
;
N13  ASSIGN,SOURCE=14,
                CURRENT_NODE=14;
ACT,,,COMP;
;
N14  ASSIGN,SOURCE=15,
                CURRENT_NODE=15;
ACT,,,COMP;
;
N15  ASSIGN,SOURCE=16,
                CURRENT_NODE=16;
ACT,,,COMP;
;
N16  ASSIGN,SOURCE=17,
                CURRENT_NODE=17;
ACT,,,COMP;
;
N17  ASSIGN,SOURCE=18,
                CURRENT_NODE=18;
ACT,,,COMP;
;
N18  ASSIGN,SOURCE=19,
                CURRENT_NODE=19;
ACT,,,COMP;
;
N19  ASSIGN,SOURCE=20,
```

```
                CURRENT_NODE=20;
    ACT,,,COMP;
;
N20  ASSIGN,SOURCE=21,
        CURRENT_NODE=21;
    ACT,,,COMP;
;
N21  ASSIGN,SOURCE=22,
        CURRENT_NODE=22;
    ACT,,,COMP;
;
N22  ASSIGN,SOURCE=23,
        CURRENT_NODE=23;
    ACT,,,COMP;
;
N23  ASSIGN,SOURCE=24,
        CURRENT_NODE=24;
    ACT,,,COMP;
;
N24  ASSIGN,SOURCE=25,
        CURRENT_NODE=25;
    ACT,,,COMP;
;
N25  ASSIGN,SOURCE=26,
        CURRENT_NODE=26;
    ACT,,,COMP;
;
N26  ASSIGN,SOURCE=27,
```

```

                CURRENT_NODE=27;
ACT,,,COMP;
;
N27  ASSIGN,SOURCE=28,
        CURRENT_NODE=28;
ACT,,,COMP;
;
N28  ASSIGN,SOURCE=29,
        CURRENT_NODE=29;
ACT,,,COMP;
;
N29  ASSIGN,SOURCE=30,
        CURRENT_NODE=30;
ACT,,,COMP;
;
N30  ASSIGN,SOURCE=31,
        CURRENT_NODE=31;
ACT,,,COMP;
;
N31  ASSIGN,SOURCE=32,
        CURRENT_NODE=32;
ACT,,,COMP;
;
; WAIT FOR PROCESSOR, COMPUTE AND SEND MESSAGES TO ALL OTHERS
;
COMP ASSIGN,AWA_FILE=SOURCE+64,
        XMT_QUE=SOURCE+96,
        XMT_ACT=SOURCE+32,

```

```

        STD_DEV=CMP_TIME*0.066667,
        TIME_RMNG=RNORM(CMP_TIME,STD_DEV,1);
BRST  AWAIT(AWA_FILE=65,96),SOURCE/1;
;
; EXECUTE A CPU BURST
;
        ASSIGN,COUNT=0,
            RECEIVER=1,1;
            ACT,,BURSTS_RMNG.EQ.1,LSTB,
            ACT,,BURSTS_RMNG.GT.1,BTM;
LSTB  ASSIGN,BURST_TIME=TIME_RMNG;
            ACT,,,GA;
BTM   ASSIGN,MEAN_BRST_LEN=TIME_RMNG/BURSTS_RMNG,
            STD_DEV=MEAN_BRST_LEN*0.06667,
            BURST_TIME=RNORM(MEAN_BRST_LEN,STD_DEV,2),1;
            ACT,,BURST_TIME.GE.TIME_RMNG,BTM;
            ACT,,BURST_TIME.LT.TIME_RMNG,GA;
GA    GOON;
            ACT,BURST_TIME;
            ASSIGN,TIME_RMNG=TIME_RMNG-BURST_TIME,
                BURSTS_RMNG=BURSTS_RMNG-1,1;
            ACT,,NPROCS.EQ.1,STOP;
            ACT,,NPROCS.GT.1,CNT;
CNT   ASSIGN,COUNT=COUNT+1,
            LAST_MSG=USERF(3);
DES   ASSIGN,DESTINATION=RECEIVER,
            RECEIVER=RECEIVER+1,1;
            ACT,,SOURCE.EQ.DESTINATION,DES;

```

```

        ACT, ,, XQ;
XQ  QUEUE(XMT_QUE=97,128);
        ACT(1)/XMT_ACT=33,64,MSG_INIT;
GOON;
        ACT, ,,LAST_MSG.EQ.0,OK;
        ACT, ,,LAST_MSG.EQ.0,CNT;
        ACT, ,,LAST_MSG.EQ.1,REL;
REL  FREE,SOURCE/1;
        ACT, ,,OK;
        ACT, ,,BURSTS_RMNG.GT.0,BRST;

;
; GET NEXT NODE AND CHANNEL NUMBER
;
OK  ASSIGN,NEXT_NODE=USERF(1);
;
; PASS MESSAGE THROUGH THE NETWORK
;
XFER  QUEUE(CURRENT_NODE=1,32);
        ACT(1)/CURRENT_NODE=1,32,MSG_SIZE*XMT_TIME;
ASSIGN,PRE_FILE=NEXT_NODE+32;
PREEMPT(PRE_FILE=33,64),NEXT_NODE;
        ACT,OVERHEAD,NEXT_NODE.EQ.DESTINATION,QUIT;
        ACT,INT_TIME,NEXT_NODE.NE.DESTINATION;
FREE,NEXT_NODE;
ASSIGN,CURRENT_NODE=NEXT_NODE,
        NEXT_NODE=USERF(1);
ACT, ,,XFER;

;

```

```
; ENSURE NODES RECEIVE ALL THEIR MESSAGES
;
QUIT  FREE,NEXT_NODE;
      ASSIGN,ARRAY(1,SOURCE)=ARRAY(1,SOURCE)+1,1;
      ACT,,ARRAY(1,SOURCE).LT.MSG_COUNT,KILL;
      ACT,,ARRAY(1,SOURCE).EQ.MSG_COUNT,STOP;
;
STOP  ASSIGN,ARRAY(1,SOURCE)=0,
      NFINISHED=NFINISHED+1,1;
      ACT,,NFINISHED.LT.NPROCS,KILL;
      ACT,,NFINISHED.EQ.NPROCS,CLCT;
CLCT  COLCT,INT(1),TIME_IN_SYSTEM;
KILL  TERM;
      END;
INIT,0;
FIN;
```

This is the main program used by SLAM II which includes the function used to find the next node and channel in the sender/receiver path. The second function is the calibration function used when the model was validated.

```
PROGRAM MAIN
DIMENSION NSET(1500000)
INCLUDE 'PARAM.INC'
COMMON/SCOM1/ATRIB(MATRB), DD(MEQT), DDL(MEQT), DTNOW, II, MFA,
1MSTOP,NCLNR, NCRDR, NPRNT, NNRUN, NNSET, NTAPE, SS(MEQT),
2SSL(MEQT),TNEXT, TNOW, XX(MMXXV)
COMMON QSET(1500000)
EQUIVALENCE (NSET(1),QSET(1))
NNSET=1500000
NCRDR=5
NPRNT=6
NTAPE=7
CALL SLAM
STOP
C
END
C
C
FUNCTION USERF(I)
COMMON/SCOM1/ATRIB(100),DD(100),DDL(100),DTNOW,II,
1MFA,MS1UP,NCLNR,NCRDR,NPRNT,NNRUN,NNSET,NTAPE,
2SS(100),SSL(100),MTEXT,TNOW,XX(100)
C
INTEGER*2 CURRENT,DESTIN,PATH,NEXT,POS,MASK,DIR
```

```

        GO TO (1,2,3) I
C
C      Function used to find next node and channel based on the
C      exclusive-or operation of the node identification numbers
C      of the source and destination.
C
1      CURRENT=ATRIB(5)-1
        DESTIN=ATRIB(3)-1
        POS=0
        PATH=IIOR(CURRENT,DESTIN)
10     MASK=2**POS
        NEXT=IIAND(PATH,MASK)
        IF (NEXT .EQ. 0) THEN
            POS=POS+1
            GO TO 10
        ELSE
            DIR=IIAND(MASK,CURRENT)
            IF (DIR .EQ. 0) THEN
                USERF=CURRENT+NEXT+1
            ELSE
                USERF=CURRENT-NEXT+1
            ENDIF
        ENDIF
        ATRIB(9)=POS+1
        RETURN
C
C      Calibration function used for model validataion.
C

```

```
2   IF (XX(42).GE.4) THEN
      USERF=5+7.5*XX(42)
    ELSE
      USERF=0
    ENDIF
  RETURN
```

C

C Function used to set a flag indicating whether or not  
C the last message has been sent by a node.

C

```
3   IF (ATRI(10).EQ.XX(47)) THEN
      USERF=1
    ELSE
      USERF=0
    ENDIF
  END
```

## Appendix C. *Benchmark Results*

This appendix lists the results of the benchmark program. All times have been rounded to integer values.

<i>Rcalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
5	1	Computation	200	200	200	200	200
		Message Processing	0	0	0	0	0
		Total	200	200	200	200	200
	2	Computation	100	100	100	103	100
		Message Processing	5	5	0	2	5
		Total	105	105	100	105	105
	4	Computation	50	51	50	51	50
		Message Processing	45	74	45	54	50
		Total	95	125	95	105	100
	8	Computation	25	26	25	26	25
		Message Processing	70	119	145	69	120
		Total	95	145	170	95	145
	16	Computation	13	13	13	13	14
		Message Processing	147	147	157	147	181
		Total	160	160	170	160	195
	32	Computation	6	7	6	7	6
		Message Processing	339	333	339	368	319
		Total	345	340	345	375	325

<i>Recalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
7	1	Computation	280	280	280	280	280
		Message Processing	0	0	0	0	0
		Total	280	280	280	280	280
	2	Computation	140	140	140	140	140
		Message Processing	5	0	5	0	5
		Total	145	140	145	140	145
	4	Computation	70	70	71	70	70
		Message Processing	40	40	54	40	40
		Total	110	110	125	110	110
	8	Computation	35	35	35	35	35
		Message Processing	115	115	75	70	70
		Total	150	150	110	105	105
	16	Computation	18	18	18	18	19
		Message Processing	167	162	152	172	221
		Total	185	180	170	190	240
	32	Computation	9	9	9	9	10
		Message Processing	311	326	371	316	280
		Total	320	335	380	325	290

<i>Rccalculations</i>	<i>Number</i>		<i>Runs</i>				
	<i>of</i>	<i>Nodes</i>	1	2	3	4	5
10	1	Computation	400	400	400	400	400
		Message Processing	0	0	0	0	0
		Total	400	400	400	400	400
	2	Computation	200	200	202	200	200
		Message Processing	5	5	3	0	0
		Total	205	205	205	200	200
	4	Computation	100	101	100	100	100
		Message Processing	55	49	40	45	40
		Total	155	150	140	145	140
	8	Computation	51	51	50	51	50
		Message Processing	74	74	70	69	70
		Total	125	125	120	120	120
	16	Computation	26	25	25	25	25
		Message Processing	149	175	155	175	185
		Total	175	200	180	200	210
	32	Computation	13	13	12	13	13
		Message Processing	342	302	333	327	372
		Total	355	315	345	340	385

<i>Recalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
16	1	Computation	640	640	640	640	640
		Message Processing	0	0	0	0	0
		Total	640	640	640	640	640
	2	Computation	320	322	320	320	322
		Message Processing	0	3	5	0	3
		Total	320	325	325	320	325
	4	Computation	160	161	160	161	161
		Message Processing	40	49	50	9	54
		Total	200	210	210	170	215
	8	Computation	80	81	81	81	80
		Message Processing	120	104	59	104	110
		Total	206	185	140	185	190
	16	Computation	40	40	40	40	40
		Message Processing	120	140	145	165	190
		Total	160	180	185	205	230
	32	Computation	21	20	20	20	20
		Message Processing	299	295	355	335	325
		Total	320	315	375	355	345

<i>Recalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
20	1	Computation	800	800	800	800	800
		Message Processing	0	0	0	0	0
		Total	800	800	800	800	800
	2	Computation	403	400	403	400	400
		Message Processing	2	5	2	5	5
		Total	405	405	405	405	405
	4	Computation	203	203	203	200	201
		Message Processing	12	47	82	50	44
		Total	215	250	285	250	245
	8	Computation	101	101	101	101	100
		Message Processing	69	74	79	99	110
		Total	170	175	180	200	210
	16	Computation	51	51	51	51	50
		Message Processing	184	154	164	174	155
		Total	235	205	215	225	205
	32	Computation	25	26	26	25	25
		Message Processing	305	339	329	340	310
		Total	330	365	355	365	335

<i>Recalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
30	1	Computation	1205	1205	1205	1205	1205
		Message Processing	0	0	0	0	0
		Total	1205	1205	1205	1205	1205
	2	Computation	600	603	603	600	603
		Message Processing	5	7	2	5	2
		Total	605	610	605	605	605
	4	Computation	301	303	301	301	300
		Message Processing	54	52	39	49	45
		Total	355	355	340	350	345
	8	Computation	151	151	151	150	151
		Message Processing	114	104	104	125	74
		Total	265	255	255	275	225
	16	Computation	77	76	77	75	76
		Message Processing	138	159	143	160	139
		Total	215	235	220	235	215
	32	Computation	39	38	38	38	38
		Message Processing	336	302	362	382	417
		Total	375	310	400	420	455

<i>Recalculations</i>	<i>Number</i>	<i>of Nodes</i>	<i>Runs</i>				
			1	2	3	4	5
65	1	Computation	2620	2615	2615	2615	2610
		Message Processing	0	0	0	0	0
		Total	2620	2615	2615	2615	2610
	2	Computation	1308	1305	1308	1308	1305
		Message Processing	7	5	2	2	5
		Total	1315	1310	1310	1310	1310
	4	Computation	655	654	655	655	654
		Message Processing	45	51	50	10	46
		Total	700	705	705	665	700
	8	Computation	327	326	326	327	328
		Message Processing	73	114	69	63	72
		Total	400	440	395	390	400
	16	Computation	165	165	164	165	164
		Message Processing	160	165	191	155	151
		Total	325	330	355	320	315
	32	Computation	81	83	83	83	83
		Message Processing	306	357	317	317	327
		Total	390	440	430	400	410

<i>Recalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
90	1	Computation	3620	3620	3620	3620	3620
		Message Processing	0	0	0	0	0
		Total	3620	3620	3620	3620	3620
	2	Computation	1808	1810	1810	1810	1810
		Message Processing	2	5	0	0	5
		Total	1810	1815	1810	1810	1815
	4	Computation	905	905	905	905	905
		Message Processing	10	55	40	40	50
		Total	915	960	945	945	955
	8	Computation	453	454	453	453	453
		Message Processing	57	141	72	62	67
		Total	510	595	525	515	520
	16	Computation	227	228	227	228	228
		Message Processing	143	132	153	217	172
		Total	370	360	380	445	400
	32	Computation	115	115	115	115	115
		Message Processing	330	335	315	325	345
		Total	445	450	430	440	460

<i>Recalculations</i>	<i>Number of Nodes</i>		<i>Runs</i>				
			1	2	3	4	5
200	1	Computation	8010	8010	8010	8010	8010
		Message Processing	0	0	0	0	0
		Total	8010	8010	8010	8010	8010
	2	Computation	4005	4005	4005	4005	4002
		Message Processing	5	5	5	5	3
		Total	4010	4010	1410	1410	1405
	4	Computation	2005	2005	2005	2005	2005
		Message Processing	50	50	50	50	55
		Total	2055	2055	2055	2055	2060
	8	Computation	1005	1004	1004	1004	1003
		Message Processing	95	96	96	56	112
		Total	1100	1100	1100	1060	1115
	16	Computation	505	505	505	505	505
		Message Processing	145	160	190	155	150
		Total	650	665	695	660	655
	32	Computation	255	255	255	255	255
		Message Processing	345	320	345	335	335
		Total	600	575	600	590	590

## Appendix D. *Model Validation*

This appendix lists the simulation results obtained using the four different uniprocessor loads that are listed in Table 3. The calibration function was incorporated into the simulation for the purpose of validation. The results from the simulation and the benchmark are averaged across each uniprocessor load and cube size. A paired-difference  $t$  test was conducted using the average times.

*Simulation Results*

<i>Uniprocessor Time</i>	<i>Number of Nodes</i>	<i>Runs</i>				
		1	2	3	4	5
200	2	103.6	101.4	102.3	108.0	104.5
	4	92.9	97.9	93.9	95.4	95.7
	8	113.3	111.4	111.8	112.8	111.3
	16	185.5	181.3	183.4	183.4	184.9
	32	351.4	349.5	351.4	351.2	350.3
1205	2	606.1	603.9	604.8	610.5	607.0
	4	341.2	349.2	345.2	346.6	346.9
	8	239.0	237.0	237.4	238.4	237.0
	16	248.3	247.3	246.4	246.3	247.9
	32	382.8	380.9	382.8	382.6	381.7
3615	2	1814.0	1811.0	1812.0	1818.0	1814.0
	4	947.9	952.9	948.9	950.4	950.7
	8	540.8	538.9	539.3	540.3	538.8
	16	399.3	398.0	397.1	397.2	398.6
	32	458.3	456.4	458.2	458.0	457.2
8010	2	4009.0	4006.0	4007.0	4013.0	4009.0
	4	2016	2050.0	2017.0	2017.0	2018.0
	8	1090.0	1089.0	1089.0	1090.0	1088.0
	16	673.6	672.4	671.5	671.6	673.0
	32	595.4	593.6	595.4	595.2	594.4

*Paired-Difference t Test*

<i>Simulation Average</i>	<i>Benchmark Average</i>	<i>Difference</i>
104.0	104.0	0.0
95.2	104.0	-8.8
112.2	130.0	-17.8
184.3	169.0	15.3
350.8	348.0	2.8
606.5	606.0	0.5
346.4	349.0	-2.6
237.8	255.0	-17.2
247.2	224.0	23.2
382.2	398.0	-15.8
1813.8	1812.0	1.8
950.1	944.0	6.1
539.6	533.0	6.6
398.0	391.0	7.0
457.6	445.0	12.6
4008.8	4009.0	-0.2
2047.6	2056.0	- 8.4
1089.2	1095.0	-5.8
672.4	665.0	7.4
591.8	591.0	3.8

$$\bar{d} = 0.525$$

$$S_d = 10.73$$

$$t = \frac{\bar{d}-0}{S_d/\sqrt{20}} = 0.219$$

## Appendix E. *Simulation Results*

This appendix lists the total execution times obtained from the simulator for each experimental unit listed in Table 5.

### Total Execution Time

One Burst at Uniprocessor Time of 1000 Milliseconds

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>
2	1	1	512.0
		2	490.1
		3	494.3
		4	569.8
		5	523.0
		6	505.3
		7	456.6
		8	544.1
		9	541.0
		10	535.0

*Number of Nodes    Number of Receivers    Run    Total Execution Time*

---

4	3	1	263.0
		2	291.9
		3	268.4
		4	279.0
		5	278.2
		6	259.0
		7	287.8
		8	274.4
		9	278.9
		10	270.7

4	2	1	260.1
		2	292.2
		3	267.4
		4	276.8
		5	274.7
		6	255.5
		7	288.2
		8	272.6
		9	276.0
		10	267.5

*Number of Nodes*   *Number of Receivers*   *Run*   *Total Execution Time*

---

8	7	1	161.4
		2	153.8
		3	155.9
		4	160.1
		5	155.3
		6	154.8
		7	154.5
		8	156.1
		9	161.2
		10	160.7

8	4	1	155.7
		2	146.8
		3	147.6
		4	152.5
		5	146.1
		6	152.0
		7	148.5
		8	150.8
		9	152.1
		10	151.5

*Number of Nodes    Number of Receivers    Run    Total Execution Time*

---

16	15	1	113.2
		2	113.0
		3	112.2
		4	111.3
		5	114.2
		6	112.5
		7	112.9
		8	113.6
		9	110.8
		10	112.5
16	8	1	92.0
		2	94.9
		3	95.6
		4	93.2
		5	98.1
		6	94.4
		7	93.0
		8	97.4
		9	93.0
		10	97.0

*Number of Nodes    Number of Receivers    Run    Total Execution Time*

---

32	31	1	134.5
		2	134.8
		3	136.6
		4	133.8
		5	133.4
		6	133.9
		7	135.0
		8	134.1
		9	135.3
		10	135.3
32	16	1	94.0
		2	92.1
		3	91.9
		4	90.1
		5	89.0
		6	89.0
		7	90.5
		8	95.2
		9	90.7
		10	90.2

AD-A194 361

A PERFORMANCE STUDY OF THE HYPERCUBE ARCHITECTURE(U)  
AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL  
OF ENGINEERING C A LANMAN JUN 88 AFIT/GCS/ENG/88J-1

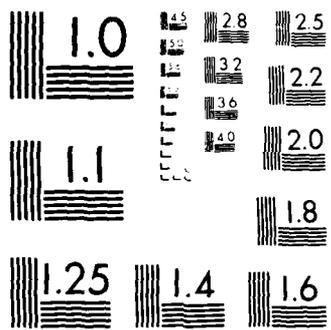
2/2

UNCLASSIFIED

F/G 12/6

NL


1  
2  
3  
4  
5



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS 1963-A

Total Execution Time

One Burst at Uniprocessor Time of 20,000 Milliseconds

*Number of Nodes*   *Number of Receivers*   *Run*   *Total Execution Time*

---

2	1	1	10170
		2	9741
		3	9814
		4	11320
		5	10390
		6	10030
		7	9048
		8	10810
		9	10750
		10	10630

*Number of Nodes*   *Number of Receivers*   *Run*   *Total Execution Time*

---

4	3	1	5091
		2	5669
		3	5201
		4	5411
		5	5382
		6	5012
		7	5587
		8	5320
		9	5383
		10	5219

4	2	1	5089
		2	5668
		3	5198
		4	5410
		5	5378
		6	5008
		7	5588
		8	5318
		9	5380
		10	5216

*Number of Nodes*   *Number of Receivers*   *Run*   *Total Execution Time*

---

8	7	1	2850
		2	2721
		3	2708
		4	2809
		5	2709
		6	2749
		7	2713
		8	2740
		9	2802
		10	2849

8	4	1	2844
		2	2713
		3	2699
		4	2800
		5	2698
		6	2743
		7	2704
		8	2732
		9	2793
		10	2840

*Number of Nodes    Number of Receivers    Run    Total Execution Time*

---

16	15	1	1460
		2	1439
		3	1408
		4	1404
		5	1458
		6	1391
		7	1453
		8	1466
		9	1411
		10	1445

16	8	1	1437
		2	1423
		3	1390
		4	1382
		5	1438
		6	1376
		7	1433
		8	1447
		9	1392
		10	1430

*Number of Nodes    Number of Receivers    Run    Total Execution Time*

---

32	31	1	805
		2	779
		3	808
		4	807
		5	795
		6	795
		7	818
		8	834
		9	814
		10	794
32	16	1	764
		2	737
		3	762
		4	763
		5	756
		6	748
		7	768
		8	795
		9	768
		10	749

Total Execution Time  
Five Bursts at Uniprocessor Time of 1000 Milliseconds

<i>Number of Nodes</i>	<i>Number Of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
2	1	1	521.1	521.1
		2	499.2	499.2
		3	503.5	503.5
		4	578.9	578.9
		5	532.1	532.1
		6	514.4	514.4
		7	465.1	465.1
		8	553.2	553.2
		9	550.1	550.1
		10	541.1	541.1

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Total Execution Time</i>		
		<i>Run</i>	<i>Normally</i>	<i>Exponentially</i>
			<i>Distributed Burst Lengths</i>	<i>Distributed Burst Lengths</i>
4	3	1	292.3	292.3
		2	321.2	321.2
		3	297.8	297.8
		4	308.3	308.3
		5	307.6	307.6
		6	288.3	288.3
		7	317.1	317.1
		8	303.7	303.7
		9	308.2	308.2
		10	300.1	300.1
4	2	1	281.7	279.8
		2	306.2	307.8
		3	285.1	288.5
		4	295.8	297.0
		5	293.8	298.0
		6	276.5	276.9
		7	305.2	304.6
		8	289.4	289.5
		9	295.4	294.3
		10	284.9	287.9

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
			8	7
		2	227.4	227.4
		3	229.6	229.6
		4	233.8	233.8
		5	229.0	229.0
		6	228.5	228.5
		7	228.1	228.1
		8	229.7	229.7
		9	234.8	234.8
		10	234.4	234.4
8	4	1	204.8	199.6
		2	188.7	184.9
		3	186.5	187.9
		4	192.1	194.8
		5	191.7	187.9
		6	193.1	194.5
		7	188.0	187.1
		8	189.0	186.6
		9	191.0	194.6
		10	197.6	200.5

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
16	15	1	283.2	283.2
		2	283.0	283.0
		3	282.2	282.2
		4	281.3	281.3
		5	284.2	284.2
		6	282.5	282.5
		7	282,8	282,8
		8	283.6	283.6
		9	280.8	280.8
		10	282.5	282.5
16	8	1	185.2	183.6
		2	195.2	195.9
		3	186.3	191.0
		4	190.3	187.8
		5	190.0	186.4
		6	188.8	192.5
		7	184.8	188.4
		8	188.9	191.7
		9	187.4	185.5
		10	186.0	185.8

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
32	31	1	512.6	512.6
		2	513.5	513.5
		3	516.4	516.4
		4	513.0	513.0
		5	511.0	511.0
		6	512.6	512.6
		7	513.1	513.1
		8	512.3	512.3
		9	514.8	514.8
		10	197.6	513.4
32	16	1	292.8	288.3
		2	291.5	296.1
		3	293.2	295.4
		4	291.9	296.9
		5	285.9	294.2
		6	290.0	292.4
		7	286.5	290.3
		8	290.8	295.7
		9	299.2	289.6
		10	293.9	298.5

Total Execution Time

Five Bursts at Uniprocessor Time of 20,000 Milliseconds

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
2	1	1	10180	10180
		2	9750	9750
		3	9823	9823
		4	11330	11330
		5	10400	10400
		6	10040	10040
		7	9057	9057
		8	10820	10820
		9	10760	10760
		10	10640	10640

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
4	3	1	5120	5120
		2	5698	5698
		3	5230	5230
		4	5441	5441
		5	5411	5411
		6	5041	5041
		7	5617	5617
		8	5349	5349
		9	5412	5412
		10	5249	5249
4	2	1	5110	5108
		2	5684	5681
		3	5217	5220
		4	5429	5429
		5	5397	5401
		6	5029	5030
		7	5603	5604
		8	5334	5337
		9	5399	5397
		10	5232	5234

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally Distributed Burst Lengths</i>	<i>Exponentially Distributed Burst Lengths</i>
			8	7
		2	2795	2795
		3	2782	2782
		4	2883	2883
		5	2783	2783
		6	2822	2822
		7	2787	2787
		8	2814	2814
		9	2876	2876
		10	2923	2923
8	4	1	2889	2886
		2	2754	2753
		3	2736	2738
		4	2844	2842
		5	2737	2738
		6	2787	2786
		7	2742	2743
		8	2769	2769
		9	2832	2835
		10	2884	2886

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Total Execution Time</i>		
		<i>Run</i>	<i>Normally</i>	<i>Exponentially</i>
			<i>Distributed Burst Lengths</i>	<i>Distributed Burst Lengths</i>
16	15	1	1630	1630
		2	1609	1609
		3	1578	1578
		4	1574	1574
		5	1628	1628
		6	1561	1561
		7	1623	1623
		8	1636	1636
		9	1581	1581
		10	1615	1615
16	8	1	1527	1524
		2	1518	1517
		3	1473	1482
		4	1479	1487
		5	1530	1527
		6	1466	1467
		7	1521	1526
		8	1539	1540
		9	1485	1482
		10	1520	1517

<i>Number of Nodes</i>	<i>Number of Receivers</i>	<i>Run</i>	<i>Total Execution Time</i>	
			<i>Normally</i>	<i>Exponentially</i>
			<i>Distributed Burst Lengths</i>	<i>Distributed Burst Lengths</i>
32	31	1	1183	1183
		2	1157	1157
		3	1186	1186
		4	1185	1185
		5	1173	1173
		6	1173	1173
		7	1196	1196
		8	1212	1212
		9	1192	1192
		10	1172	1172
32	16	1	960	965
		2	943	938
		3	962	966
		4	957	955
		5	951	951
		6	951	942
		7	965	964
		8	993	995
		9	963	967
		10	948	951

## Bibliography

- [FE85] Felten, Edward *et al.* "The Traveling Salesman Problem on a Hypercube, MIMD Computer," *Proceedings of the 1985 International Conference on Parallel Processing*, 6-10. Washington, DC: IEEE Computer Society Press, 1985.
- [GU87] Gutzmann, Kurt M. "Optimal Dimension of Hypercubes for Sorting," *Computer Architecture News*, 15: 68-72 (March 1987).
- [HW84] Hwang, Kai and Fáyé A. Briggs. *Computer Architecture and Parallel Processing*. New York: McGraw Hill Book Company, 1984.
- [IN86a] Indurkha, Bipin *et al.* "Optimal Partitioning of Randomly Generated Distribution Programs," *IEEE Transactions on Software Engineering*, SE-12: 483-495 (March 1986).
- [IN86b] Intel Corporation. *iPSC Software Internal Product Specification*. Beaverton, OR, April 1986.
- [LE86] LeBlanc, Thomas J. "Shared Memory Versus Message Passing in a Tightly-Coupled Multiprocessor: A Case Study," *Proceedings of the 1986 International Conference on Parallel Processing*, 463-466. New York: IEEE Press, 1986.
- [MO87] Moore, Timothy S. *A Simulation Study of a Parallel Processor with Unbalanced Loads*. MS thesis, AFIT/GSC/ENG/87D-20, Air Force Institute of Technology (AU), Wright Patterson Air Force Base, OH, December 1987.
- [NI87a] Ni, Lionel M. *et al.* "Parallel Algorithm Design Considerations for Hypercube Multiprocessors," *Proceedings of the 1987 International Conference on Parallel Processing*, 717-720. University Park, PA: The Pennsylvania State University Press, 1987.
- [NI87b] Nicol, David M. and Frank H. Willard. "Problem Size, Parallel Architecture, and Optimal Speedup," *Proceedings of the 1987 International Conference on Parallel Processing*, 347-354. University Park, PA: The Pennsylvania State University Press, 1987.
- [PR86] Pritsker, A. Alan B. *Introduction to Simulation and SLAM II*. New York: John Wiley and Sons, 1986.
- [RE88] Reed, Daniel A., Assistant Professor. Telephone interview. Department of Computer Science, University of Illinois at Urbana-Champaign, 3 May 1988.
- [RE87] Reed, Daniel A. and Dirk C. Grunwald. "The Performance of Multicomputer Interconnection Networks," *Computer*, 20: 63-73 (June 1987).
- [ST87] Stone, Harold S. *High Performance Computer Architecture*. Reading, MA: Addison-Wesley Publishing Company, 1987.

- [SE85] Seitz, Charles L. "The Cosmic Cube," *Communications of the ACM*, 28: 22-33 (January 1985).
- [WI87] Wiley, Paul. "A Parallel Architecture Comes of Age at Last," *IEEE Spectrum*, 24: 46-50 (June 1987).

## *Vita*

Captain Catherine Lamanna was born on 23 July 1957 in Kenosha, Wisconsin. She graduated from Hanahan Senior High, Hanahan, South Carolina, in 1974, and shortly thereafter, enlisted in the Army. In 1981, she left the Army and entered Utica College of Syracuse University where she graduated with a Bachelor of Science Degree in Computer Science. Upon graduation, she received her commission in the Regular Army as a Signal Corps Officer and returned to active duty in May of 1983. She served for three years in the 50th Signal Battalion at Fort Bragg, North Carolina as a platoon leader and later, the Battalion Adjutant. Captain Lamanna entered the School of Engineering, Air Force Institute of Technology, in October of 1986.

Permanent address: Route 5, Box 108A  
Asheboro, North Carolina  
27203

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited	
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/ENG/88J-1		7a. NAME OF MONITORING ORGANIZATION	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (if applicable) AFIT/ENG	7b. ADDRESS (City, State, and ZIP Code)	
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, OH 45433-6583		9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8a. NAME OF FUNDING / SPONSORING ORGANIZATION OSD/SDIO	8b. OFFICE SYMBOL (if applicable)	10. SOURCE OF FUNDING NUMBERS	
8c. ADDRESS (City, State, and ZIP Code) Pentagon ATTN: S/BM Washington, D. C. 20301-7100		PROGRAM ELEMENT NO.	PROJECT NO.
		TASK NO.	WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) A PERFORMANCE STUDY OF THE HYPERCUBE ARCHITECTURE			
12. PERSONAL AUTHOR(S) Catherine A. Lamanna, Captain, USA			
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) June 1988	15. PAGE COUNT 116
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD	GROUP	Parallel Processors	
12	06	Computer Architecture	
		Computer Systems Simulation	
19. ABSTRACT (Continue on reverse if necessary and identify by block number) Thesis Chairman: Wade H. Shaw, Jr., Captain, USA Assistant Professor of Electrical and Computer Engineering			
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT <input type="checkbox"/> UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Wade H. Shaw, Jr., Captain, USA		22b. TELEPHONE (Include Area Code) (513) 255-3576	22c. OFFICE SYMBOL AFIT/ENG

over

Approved for public release: IAW AFR 100-14  
Lynn E. Wolaver 15 Feb 88  
Dept for Research and Professional Development  
Air Force Institute of Technology (AFIT)  
Wright-Patterson AFB OH 45433

## *Tesis*

→ This ~~study~~ investigated the relationship between workload characteristics and process speedup. There were two goals: the first was to determine the functional relationship between workload characteristics and speedup, and the second was to show how simulation could be used to determine such a relationship. The hypercube implementation used in this study is a packet-switched network with predetermined routing. Message processing has precedence, so nodes are interrupted during task processing.

In this study, three independent variables were controlled: total computational workload, number of nodes and the message traffic load. The workload was assumed to be balanced across the nodes. A benchmark program was executed on an actual hypercube and the results were used to validate a discrete event simulation model of hypercube processing. Using the simulation, an experiment was designed to control the total computational load over two levels, the number of nodes over five levels and the message traffic load over four levels to determine their individual and interactive effects on process speedup. ←

Regression analysis was used to estimate the functional relationship between the three independent variables and process speedup. The results show that a complex relationship exists between workload characteristics and cube size. As more nodes are added, the computational time decreases, but at the same time, the communications overhead increases such that the speedup will eventually begin to decrease. The point where speedup starts to decline is dependent upon both the computational and message traffic workload. Finally, this research presented an alternative methodology for performance analysis which is more flexible than the traditional methods. Furthermore, this methodology can be extended to study other architectures.

END

DATE

FILMED

9-88

DTIC