RESOURCE SCHEDULING ON INTERCONNECTION NETWORKS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Anthony Hoai Hicks

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical Engineering

August 1982

This thesis is dedicated to my parents

and my little sister.

ACKNOWLEDGMENTS

I would like to give special thanks to my advisor, Professor Benjamin Wah for his guidances, encouragement, patience, and constructive criticism during the research of this thesis. In addition, I also would like to acknowledge the support for this research from the National Science Foundation by Grant ECS 80-16580.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Hicks, Anthony Hoai, M.S.E.E., Purdue University, August 1982. Resource Scheduling on Interconnection Networks. Major Professor: Benjamin Wah.

In this thesis, problems of resources scheduling using interconnection networks are studied. Emphasis is placed on the distributed control of routing an incoming request to a pool of free resources. This differs from the conventional address mapping technique which routes requests to a given destination. A distributed control algorithm is designed for this purpose. To establish a bench march for performance comparison, an optimal centralized algorithm is also studied. It requires a network controller which becomes the bottleneck as the size of network increases. Other scheduling heuristics are also compared. The centralized blocking probabilities are virtually zero, while blocking probabilities for other of algorithms are less than 20% in all cases.

CHAPTER I - INTRODUCTION

The new and rapid advances in semiconductor technology has given way to the ability to pack an incredible amount of computing power into very tiny size processing units. Day after day, new applications have been popping up which require higher computing speed, and have led up to the development of parallel processing.

In a parallel processing system which consists of numerous general or special purpose processing units, an effective mean of communication among the processors and the resources is needed. It is very expensive to directly connect each of those processing units to all available resources, especially when the number of processing units is large. A interconnection network is a good solution to this problem. Its function is to route requests from some input ports to some output ports. Examples of these networks include the Banyan[GOK73], Binary n-Cube[PEA77], Cube[SIE81a], Perfect Shuffle[STO71], Flip[BAT76], Omega[LAW75], Data Manipulator[FEN74], Augmented Data Manipulator (ADM)[SIE81b], Delta[PAT81], and Baseline[WU80]. Examples of systems designed with interconnection networks are TRAC [SEJ80], STARAN [BAT74], C.mmp [WUL72], Numerical Aerodynamic Simulation Facility (NASF) [BAR81, BUR79] and the Ballistic Missile Defense testbed [MCD78].

1.1. Statement of the Problem

For the purpose of routing requests from a set of processing units to a pool of resources (Figure 1.1), the interconnection network is termed a Resource Sharing Interconnection Network (RSIN). Resource sharing means that all the available resources are shared among requesting processors. In this respect, requests initiated from source processors can be sent to any one of the free resources of a given type at the destination. This is the main difference that differentiates RSIN's from conventional interconnection networks which route requests to a given destination.

As an illustration, consider a 4 by 4 Omega network (Figure 1.2). Assume processors 0, 1, 2 are making requests and resources 0, 1, 2 are available, Processor 3 is not making a request and Resource 3 is busy. Further, the network is completely free. All the resources will be allocated if the following processor-resource mappings are used: {(0,0), (1,1), (2,2)}, {(0,1), (1,0), (2,2)}, {(0,2), (1,0),(2,1)} or {(0,2), (1,1), (2,0)}. But if the following processor-resource mappings are used: {(0,0), (1,2), (2,1)} or {(0,1), (1,2), (2,0)}, then a maximum of 2 resources can be allocated without blocking. This gives a resource utilization of 66%. A similar example can be generated for the Indirect Binary n-Cube network. This shows that the scheduler must be designed properly to give the maximum resource utilization.

The earliest study of RSIN has been done with respect to centralized computer systems. A uni-bus is used in a time shared fashion for connecting peripheral I/O devices to the CPU. Multiple time-shared buses have been used in the PLURIBUS minicomputer multiprocessor [ORN75]. A Cross-bar switch has been used in C.mmp [WUL72]. This network permits full interconnection capability between source and destination ports. As long as each source port

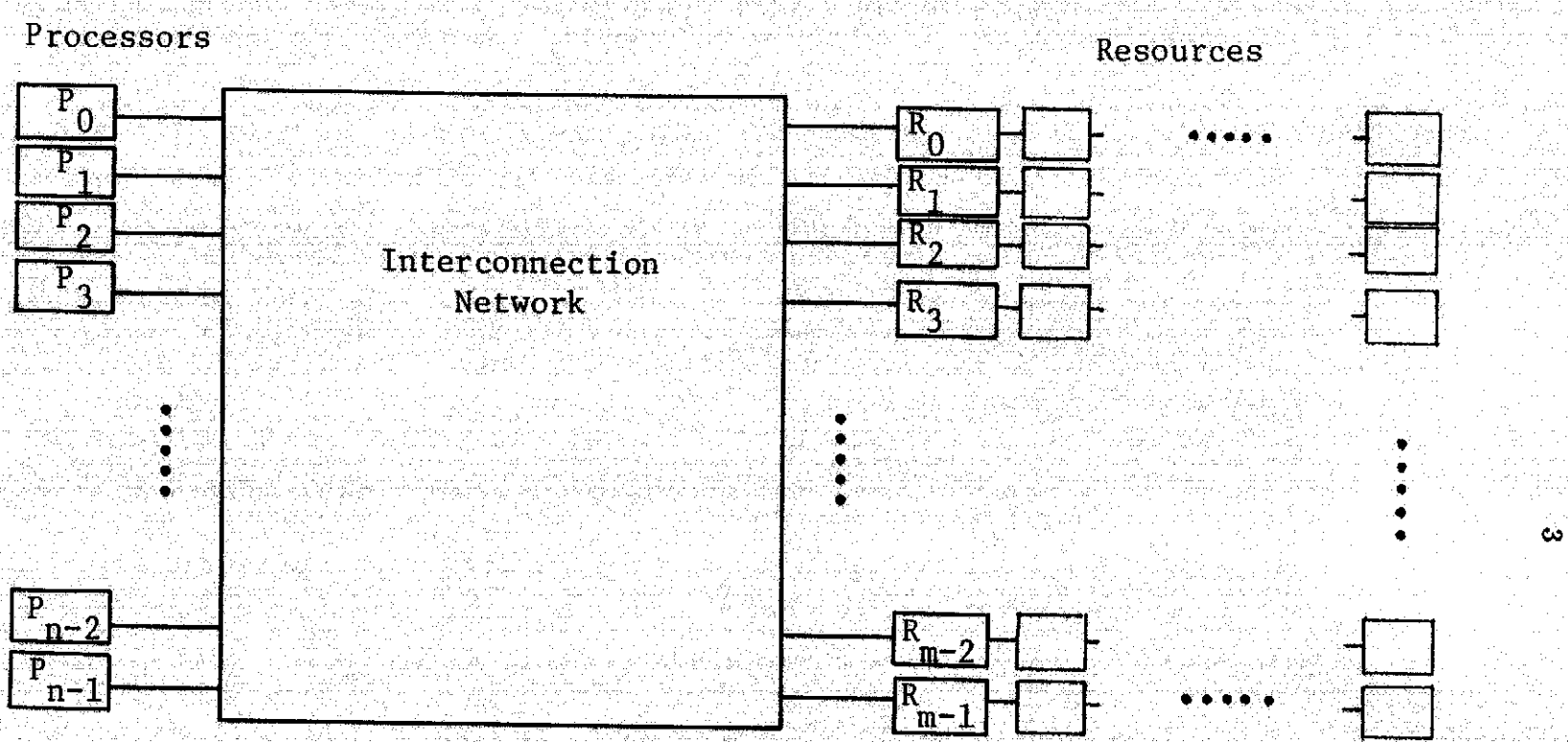
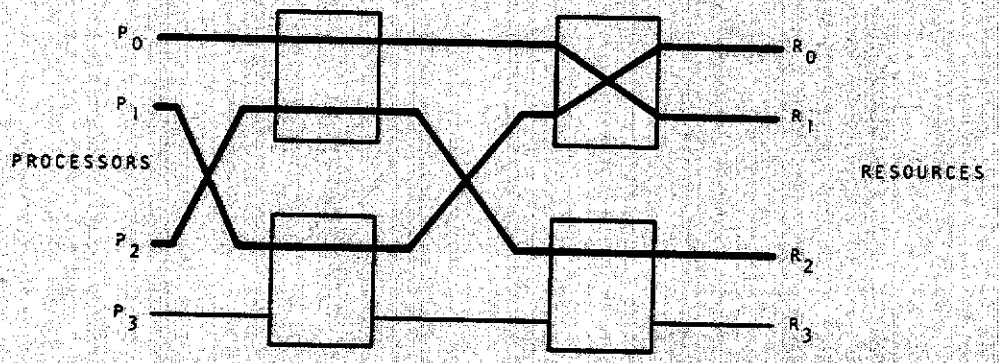
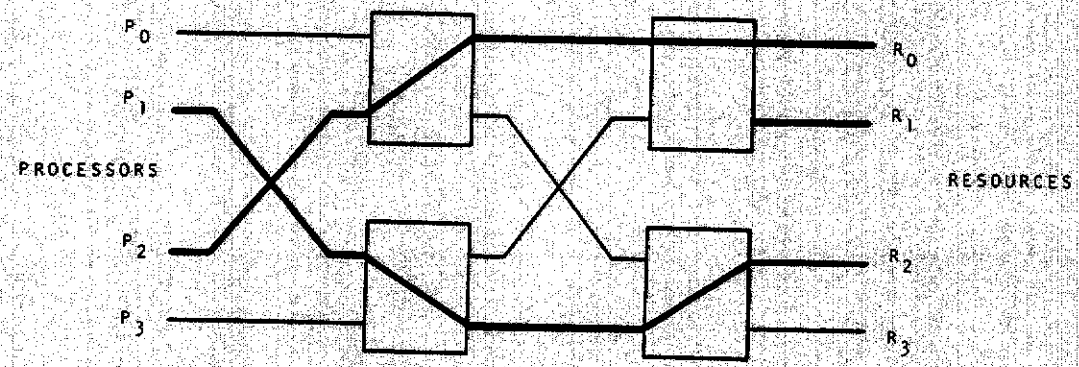


Figure 1.1 - A Resource Sharing Interconnection Network



(a) Processor-Resource mapping {(0,1), (1,0), (2,2)}
All resources are allocated.



(b) Processor-Resource mapping {(0,1), (1,2), (2,0)}
Only 2 of the resources are allocated.

Figure 1.2 - A RSIN using 4x4 Omega Network.

addresses a unique destination port, there is no blocking in the network and all messages can be routed through the network simultaneously. The single or multiple buses is a source of bottleneck, and is the least expensive design. The Cross-bar switch is the most expensive networks, but has the least degree of blocking. A compromise is to use a less expensive network than the cross-bar switch and has a lower blocking probability than the single bus systems. This has been studied with respect to the Banyan network [RATH80, JEN81]. In these studies, it is shown that when a processor makes a request for multiple resources, by allocating resources with smaller distance functions, the amount of network blockage is reduced [GOK76]. A tree network is proposed to aid the scheduler in choosing a suitable resource to allocate. The search is done in pre-order. Therefore, the tree network has a delay of O(n) in selecting a free resource (n is the total number of resources), and most notably, the scheduling of multiple requests is done sequentially.

A few comments can be made about the previous studies. First, the scheduling algorithms are centralized. For mapping n requesting processors to n resources, the scheduling algorithm has a worst case complexity of $O(n^2)$. This complexity depends on the number of requesting processors and practical when n is small or when requests are not very frequent. Second, for scheduling requests on interconnection networks with logarithmic delays such as the Binary n-Cube, Banyan and Omega, no optimal scheduling algorithm has been established.

1.2. Motivations

This study was motivated by several possible applications which may be useful in various aspects of computing such as distribution of a network load, VLSI special purpose chips, and Data Flow Multiprocessors.

1.2.1. Load balancing Network

It is possible to set up the network in such a way that a processor can request service from other idle processors. This is done by mapping the processors one-to-one onto themselves using the RSIN. In other words, the network outputs are connected directly back to the processors on the input side through the RSIN as illustrated by Figure 1.3. In this situation, any of the processors can request service from any other processors include itself through the distributed controlled network. The workload is therefore, balanced in the system.

1.2.2. VLSI Resource Pools

The requesting processors can use the RSIN to access a pool of special purpose VLSI chips performing tasks like matrix multiplication, matrix inversion, Fast-Fourier Transform, etc... (Figure 1.4) A processor may request several tasks of matrix multiplication to be done from the VLSI resource pool. The RSIN is responsible for routing a task to the available resources.

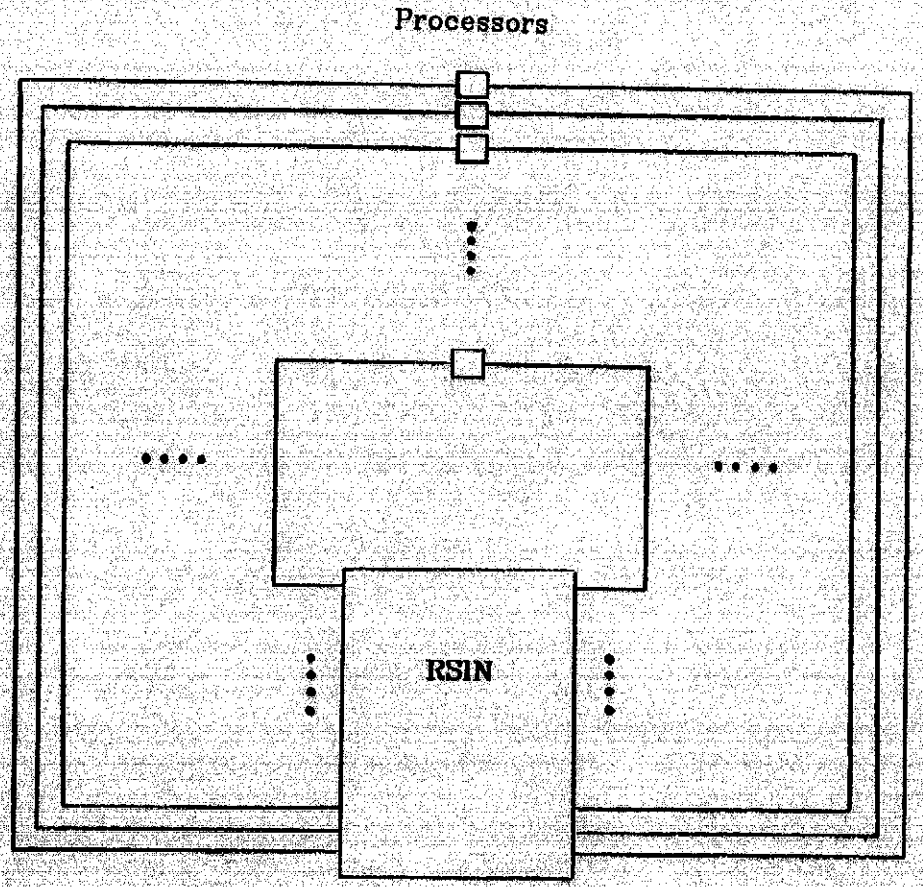


Figure 1.3 - Load Balancing Network.

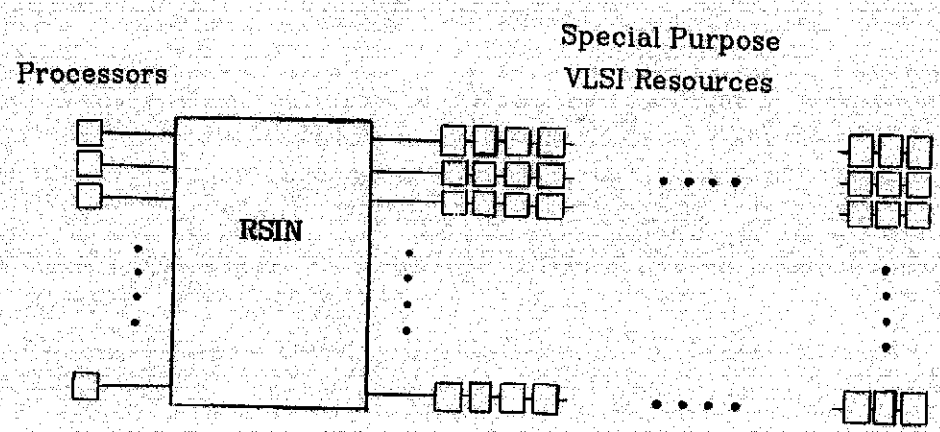


Figure 1.4 - An example of task using RSIN to schedule task to a VLSI resource pool.

1.2.3. Data flow machine.

Computational speed is becoming more and more important in today's applications. It is certainly faster if processing is done in parallel. A data flow multiprocessor seems to be a good candidate. [W&G82, RUM77] But there arises the question of efficiently utilizing of resources in the system.

It is possible to use the distributed RSIN to allocate processors to requesting processors. These processors are assumed to be general purpose. (Figure 1.5) When a processor is finished with a task, it is forwarded to any of the free memory modules through another distributed controlled network. The correlation of different tasks is not shown here. This is assumed to be done by a separate associative memory.

1.3. Objectives of the study

The objectives of this thesis are: a) To study the performance of different scheduling algorithms with an emphasis on distributed control and a scheduling time that is independent of the number of requests made. b) To design interconnection networks supporting distributed scheduling.

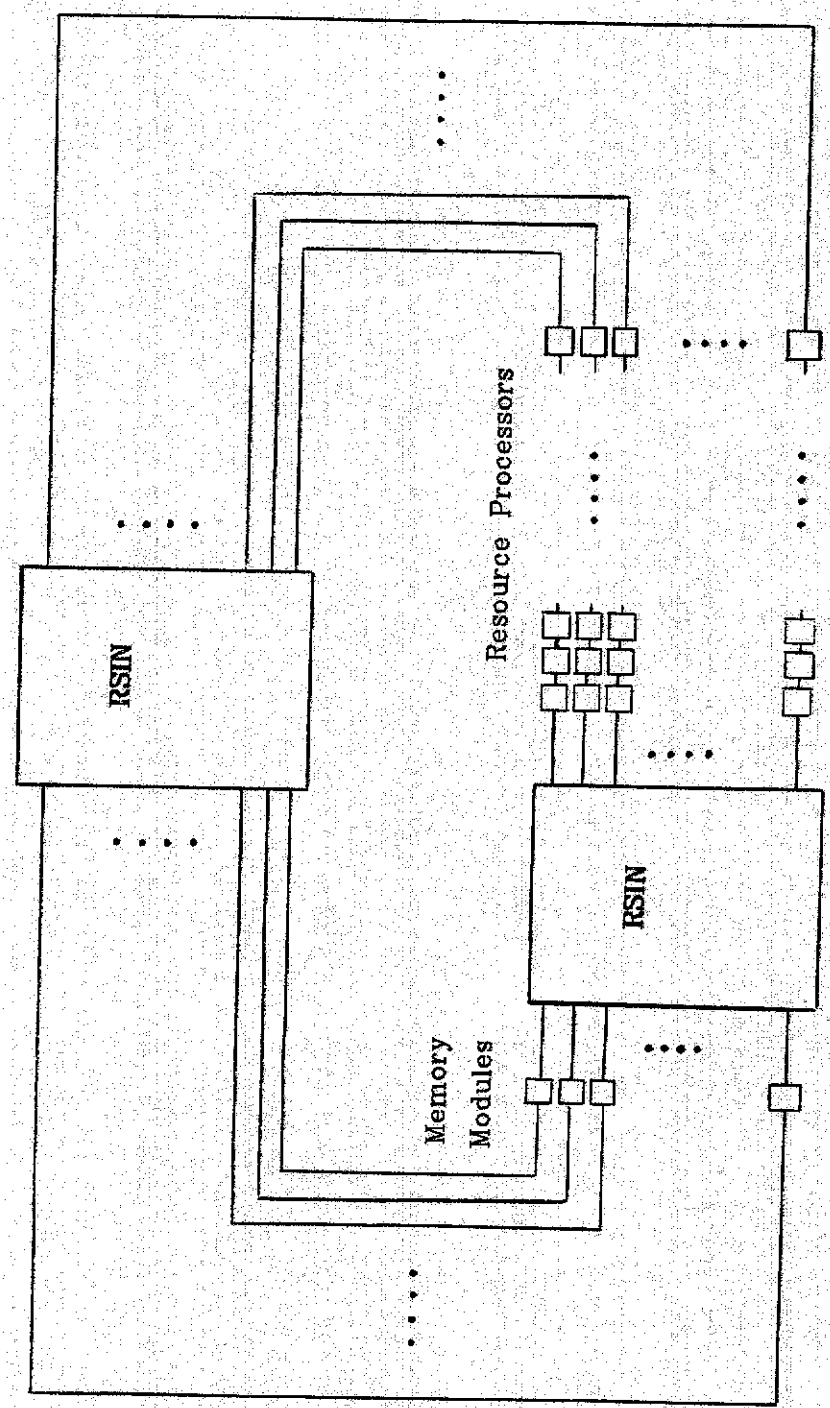


Figure 1.5 - Data Flow Multiprocessors using RSIN's.

1.4. Assumptions

The basic assumptions made when the algorithm is discussed are that each processing unit requests only one resource at a time and there is one type of resources in the system. In the simulations, the request of multiple resources is allowed. Two types of requests need to be characterized:

a) For each request, the processor has to be connected continuously to the requested resource for the duration of that request. In this case, the RSIN may not be completely free when new requests are initiated. Networks with logarithmic delays such as the Omega and Indirect Binary n-Cube may have very high blocking probability. The Crossbar switch is essential here.

b) Requests are made in block transfer mode. In this case, a resource is connected to a requesting processor for only a short duration of data transfer. The connection can be broken after the transfer while the resource is processing the data. In this case, the network is almost or completely free when new requests are initiated. In this case, networks like the Indirect Binary n-Cube and the Omega are suitable to be used.

In the presentation of the algorithms, the network is assumed to be completely free when new requests arrive. The performance of partially busy network will be investigated in the simulations.

1.5. Organization of thesis

This thesis is organized into five chapters. A brief background of the problem is given in chapter 2. In Chapters 3 and 4, the algorithms and performance results are discussed. Finally, a discussion on the direction of future research is given in Chapter 5.

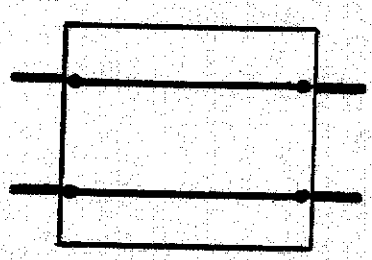
CHAPTER II - BACKGROUNDS

2.1. Definition of Interconnection Network

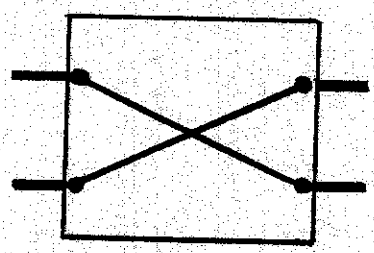
An interconnection network is a network which connects a set of input processors to a set of output resources through one or more stages of interconnected switches. A multistage network is usually made of of a matrix of control boxes. Each column of the matrix is a stage of the network. The number of stages depends on the characteristics of the network.

A control box is a 4-function interconnected switch which has two inputs and two outputs. Its functions are straight, diagonal, lower broadcast and upper broadcast connections (Figure 2.1). The 4-function exchange box is a main characteristic for the Generalized Cube network. The Indirect Binary n-Cube has only 2 functions of Straight and Diagonal setting.

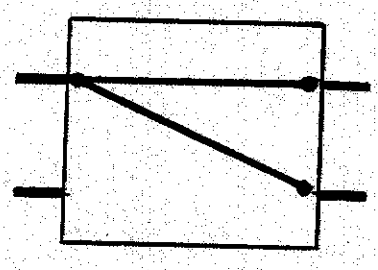
The actual connection pattern between these control boxes varies for different networks. Crossbar, Indirect Binary n-Cube and Omega networks are chosen for this study.



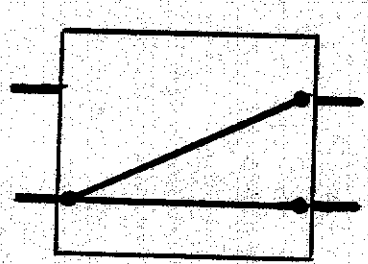
Straight



Diagonal



Upper Broadcast



Lower Broadcast

Figure 2.1 - Four-function Control Box.

2.2. Cross-bar, Binary n-Cube and Omega network

Figures 2.2, 2.3, and 2.4 describe the overall structure of the Omega, binary n- Cube and Cross-bar networks. The crossbar network permits full interconnection capability between any single input and any output. Using conventional addressing method, as long as an input port addresses a unique output port, there is no blocking in the network. In other words, all messages can be routed through the network simultaneously. Since there is no blocking in this network, only a Distributed Controlling scheme of this network will be studied in this thesis.

In this thesis, the term Binary n-Cube implies the for Indirect Binary n-Cube network. It can be described uniquely as a conceptual cube with the following properties [SIE79] (Figure 2.5):

- N input and output ports can be represented in binary numbers.

- Log_2N stages of the network is the number of binary bits used in representing the N ports.

- Horizontal line on the Binary n-Cube connects vertices which have 0^{th} (low-order) bit position differed.

- Diagonal lines connects vertices which have 1^{st} bit position differed.

- Vertical lines connects vertices which have 2^{nd} (high order) position differed.

In other words, vertices on horizontal /diagonal/vertical lines represent input lines connected to the same box at stage 0 /1/2 in the network. Therefore, the corresponding bit control the corresponding stage in the network.

An N*N Omega network consists of the same number of stages as of an N*N Cube network. Each stage consists of a Perfect Shuffle [SIE79] interconnection followed by N/2 control boxes as shown in Figure 2.2. The perfect

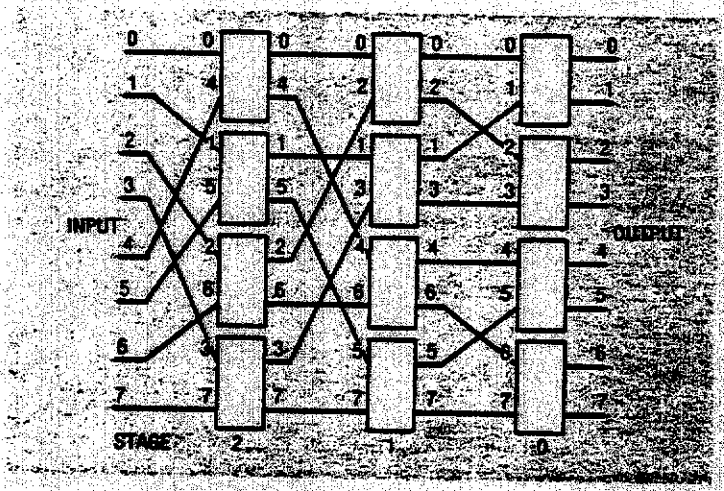


Figure 2.2 - The Omega Interconnection Network.

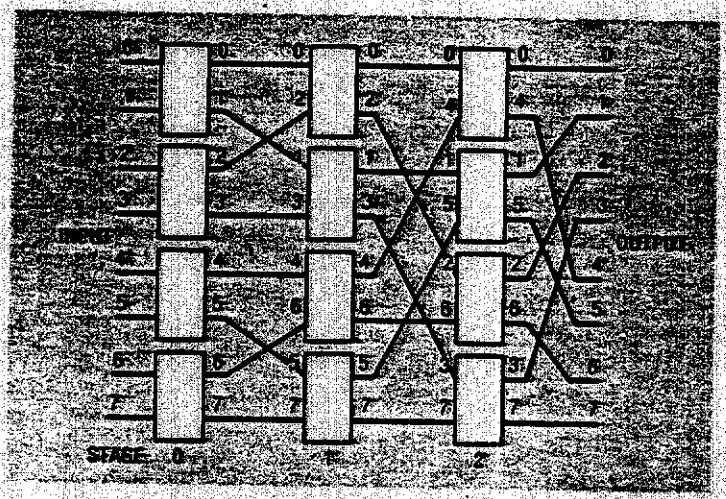


Figure 2.3 - The Indirect Binary n-Cube Network.

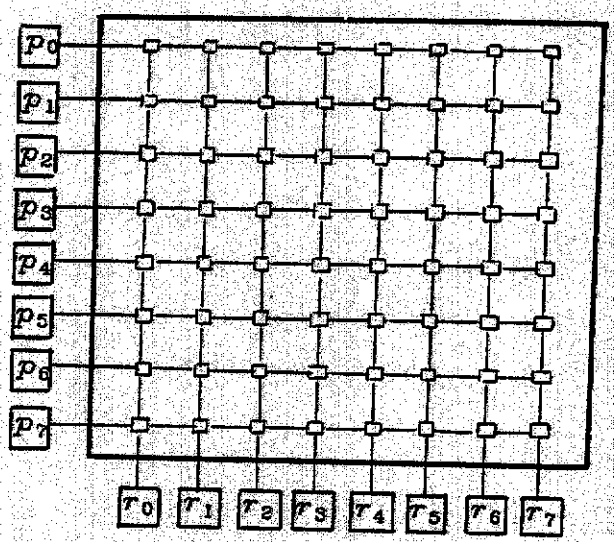


Figure 2.4 - An 8x8 Crossbar Switch.

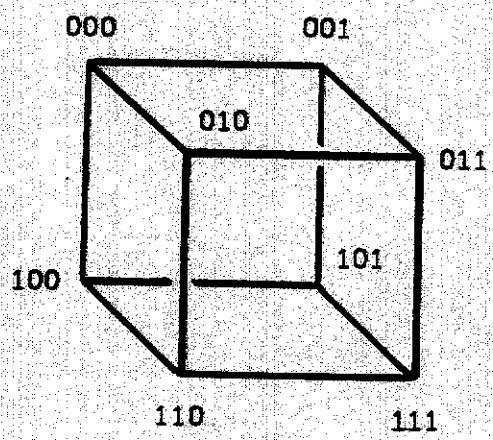


Figure 2.5 - The Cube Representation.

shuffle connection has the characteristics of taking an input at a position whose binary representation is S, S_2 . . . S_ρ and moving it to position S_2S_3 . . . $S_\rho S_1$; [LAW75]. Then the control box can move the output to S_2S_3 . . . $S_\rho 0$ or S_2S_3 . . . $S_\rho 1$. In addition; by redrawing the Omega network, we can have a Binary n-Cube network drawn backward. So we can summarize the characteristics of the Omega network as that of the Binary n-Cube network with the exception of it being backward.

In this study, due to the limitation of time and space for simulation, an 8x8 (8 inputs and 8 output) Omega and Binary n-Cube network are chosen.

2.3. Address mapping technique

In the conventional usage of interconnection networks, a central controller is required to examine the destinations of requests and the availability of destination resources in order to set the exchange boxes accordingly. Referring to a Binary n-Cube network (Figure 2.3), the controller utilizes destination addresses as a mean to control the network in the following manner:

- Assign a "0" to the top output port of an exchange box. For an entering request to an exchange box with a setting of "0", it will be directed to the top output port of exchange box.
- Assign a "1" to an bottom output port of the exchange box. A setting of "1" in the box will direct the request signal to the lower output port of that exchange box.

Since the number of stages in a Binary n-Cube network is $\log_2 N$ (N is the number of input or output ports in the network [SIE 79].), this is equal to the number of bits used to represent an output port's address in binary. These bits in an address from any input of the network are used to set the corresponding box in the corresponding stage and to direct a request to the exact destination. For instance, a request wants to get to output port 6 from input port 2. Using Figure 2.3 and address of port 6 as 110 in binary, the request goes from the top input of the 2nd box in stage 0 to the bottom input of the 1st box in stage 1. Then follows the bottom output to the top input of the 3rd box in stage 2, and finally to the bottom output of that box, the request will get to output port # 6 of the network.

This method is referred as the conventional address mapping technique.

2.4. Worst-Case Computational Complexity of RSIN

To design an effective scheduling algorithm for these networks, a careful consideration is needed on the ordering of incoming requests. Using the illustrated example in the Introduction of this thesis, blocking can be minimized by reordering the connections. Therefore, all the permutations between input and output ports should be considered. Let $C\binom{N}{M}$ is the combination of N choosing M. There are:

$$\sum_{i=1}^{N} \binom{N}{i} \text{ Combinations of input ports}$$

which can make requests for data transfer through the network.

Further, there are:

$$\sum_{j=1}^{N} \binom{8}{j}$$

possible combinations of resources being available. For each combination of requests and resources, we need to go through all the possible matching between processors and resources in order to find the maximum resource utilization. This is done in the following way. Suppose there are i processors making requests and j resources are available and $i \geq j$ From the i processors, we selected i of them to allow them to be serviced. The j processors selected must be ordered with j! ways in order to exhaust all the permutations. Therefore, there are:

$$\binom{i}{j} j! \qquad i \geq j$$

$$\binom{j}{i} i! \qquad i < j$$

ways of ordering these i processors and j resources. The worst case complexity of a RSIN is,

$$\sum_{i=1}^{N} \sum_{j=1}^{i} \binom{8}{i} \binom{8}{j} \binom{i}{j} j! \quad + \quad \sum_{j=2}^{N} \sum_{i=1}^{j} \binom{8}{i} \binom{8}{j} \binom{j}{i} i!$$

For N=8, this expression is evaluated to be 8,546,176 which is quite too large for any simulation method. In other words, it is totally infeasible to examine all the possible permutations using the computer in measuring the

performance of a particular control scheme for a RSIN. In consequence, a better approach was developed by finding the network's proper constraints which in turn eliminates all redundant cases of permutation. This is shown in the next chapter.

2.5. Blocking probability in resources allocation

As illustrated in chapter I, blocking can be minimized most of the time if a set of incoming requests is permuted correctly to a set of available resources. Since blocking probability is so important in this study, it is identified into two types: processor blocking probability and resource blocking probability.

Processor blocking probability is defined as:

$$\text{Processor blocking probability} = \frac{\text{Number of allocated processors}}{\text{Number of requesting processors}}$$

Average processor blocking probability is the average taking for all sets of request which have the same number of requesting processors and the same number of available resources.

Resource blocking probability is computed in a similar manner.

In this study, average processor blocking probability will be used and must be interpreted correctly. For instance, if there are 4 processors requesting and 2 resources available, then at most (without blocking) 2 processors can be allocated. We can say that the minimum processor blocking probability is .5 and the minimum resource blocking probability is 0.

CHAPTER III - SCHEDULING ALGORITHMS FOR RESOURCE SHARING.

3.1. Centralized schedule algorithm:

In this section, we will discuss the design of centralized scheduling algorithms and heuristics for the Crossbar, Binary n-Cube, and Omega networks.

3.1.1. Cross-bar Network Design

As mentioned in the previous chapter, work has been done on centralized scheduling for the cross-bar switch in C.mmp[WUL72]. This network allows full interconnection ability from any input processor to any output resource as long as the processor addresses a unique output port. In other words, there is no blocking in the network when a centralized scheduling scheme is used. Since there are N inputs and N outputs, we have N^2 control boxes which results in the time complexity of controlling this network of $O(N^2)$. In comparison with logarithmic delayed networks, there is certainly more hardware involved. This

network is very efficient and feasible when N is small. A design for this will not be discussed here.

3.1.2. Logarithmic-Delay-Networks Design

3.1.2.1. Optimal Algorithm

As shown in the last chapter it is totally infeasible to approach the problem using an enumerative method. In the following sections, a different approach will be discussed.

3.1.2.1.1. Table of connections: For the Binary n-Cube or Omega network, let N be the number of input and output ports and n represents the number of stages in the network. For the particular case of N=8, we have n=3.

Every stage has N/2 boxes, resulting in a total of (N/2)n boxes

For an 8x8 Binary n-Cube or Omega network, there are :

$$\frac{8}{2}\log_2 8 = 12 \text{ boxes}$$

Let's assume that every box has only two (5 states (straight or diagonal setting) for the time being. This can be represented as a binary variable. With 12 binary variables, there are $2^{12} = 4096$ possible states for this 8x8 network.

In other words, there are 4096 unique ways that 8 input ports can be simultaneously connected to 8 output ports without blocking. A table of connections between input and output ports was generated which has 4096 entries describing the 4096 unique connections. This figure seems to be more reasonable than the 8.5 million evaluated earlier.

All redundancies of permutations are eliminated from the 4096-connection table. This table makes it possible for some searching schemes such as hashing.

Because of the nature of the problem, it requires individual examination of each connection between the set of requesting input ports and the set of available output ports to finalize a best set of connections. For instance, the table contains 4096 entries of 8 values each. In other word it has 8 columns and 4096 rows. Each column is assigned to an input port. Each value in every row represents an output port, and each row represents (Figure 3.1) a set of connections between the 8 input ports and 8 output ports without blocking. For example, from the 6^{th} row of the table, input and output port connections are: 1-1, 2-6, 3-3, 4-8, 5-5, 6-2, 7-7, 8-4. If there are a set of requesting input ports (2,4,7), and a set of available output ports (2,3,6), the procedure to find a best set of connections is as followed:

1. Column 2, 4 and 7 are searched.

2. Entry numbers are picked out and retained for those which has output ports values 2, 3, 6.This is done separately for all 3 columns.

3. Intersection needed to be done to get a best set of connection.

In the case that not all three of the output ports are available then a set of two out of three processors have to be examined each time using the same procedure described. This brings back in the permutation problem. A similar

Table of Connections

Processor:	1 2 3 4 5 6 7 8
Entries:	**Resource**
1 :	1 2 3 4 5 6 7 8
2 :	1 2 3 8 5 6 7 4
3 :	1 2 7 4 5 6 3 8
4 :	1 2 7 8 5 6 3 4
5 :	1 6 3 4 5 2 7 8
6 :	1 6 3 8 5 2 7 4
.	. .
.	. .
.	. .
.	. .
.	. .
4096 :	8 6 7 4 5 2 3 1

Figure 5.1 - Example of a Table of Connections for a 6*8 Binary n-Cube network.

problem happens if there are more available output ports than requesting input ports.

The above procedure is still not feasible to be simulated using the computer. Therefore, it calls for a different organization of the 4096-connections table to enable faster search time for the best connection.

3.1.2.1.2. Tree structure representation of connections table: From the 4096-connections table, there is another way of representing it. Since there are 4096 unique entries, it seems very logical to organize them a tree with 4096 unique paths and for every path, it is a depth of 8. (Figure 3.2)

In general, if input port 1 is connected to output port 1 then input port 2 is only able to connect to output ports 2, 3, 4, 5, 6, 7 and 8. Further down, input port 3 now is only able to connect to output ports 3, 4, 5, 6, 7 and 8 and so on This is one path in the 4096 paths of the table. If we repeat this for input port 1 to output 2, then 3, then 4 and so on, a complete general tree is created as the diagram described below. The only difference is that not all entries that are assumed on the tree will be in the table. It is only necessary to build the tree according to the information given in the table. Therefore, that tree is much smaller than the original one (Figure 3.3).

All siblings now are linked together in chronological order as linked lists of subtrees, with the leftmost sibling as a head node of that linked list on that level. That head node is linked as a son to the one on top of it. This representation is an unbalanced tilted binary tree which enables us to do traversing

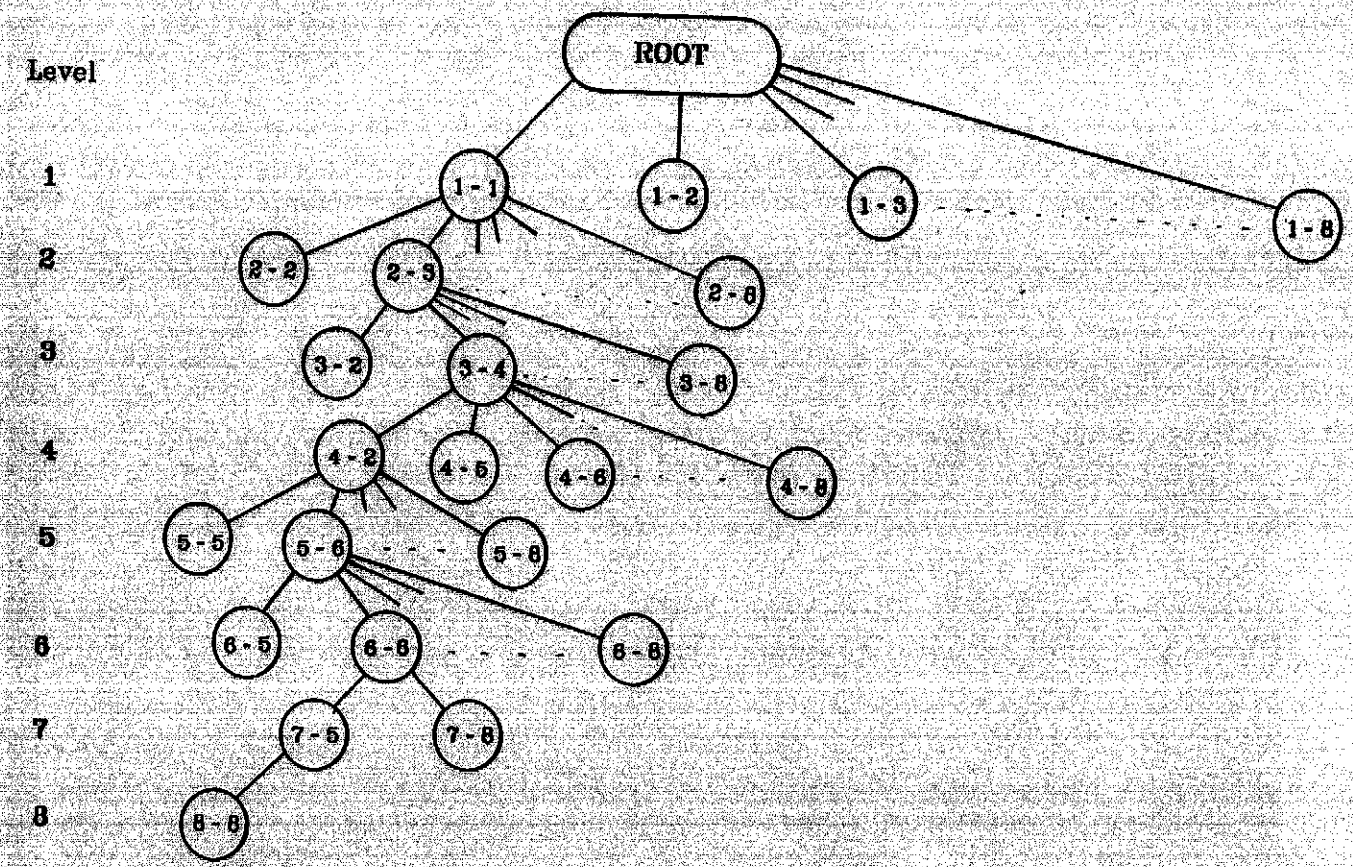


Figure 3.2 - The Generalized Tree Representation.

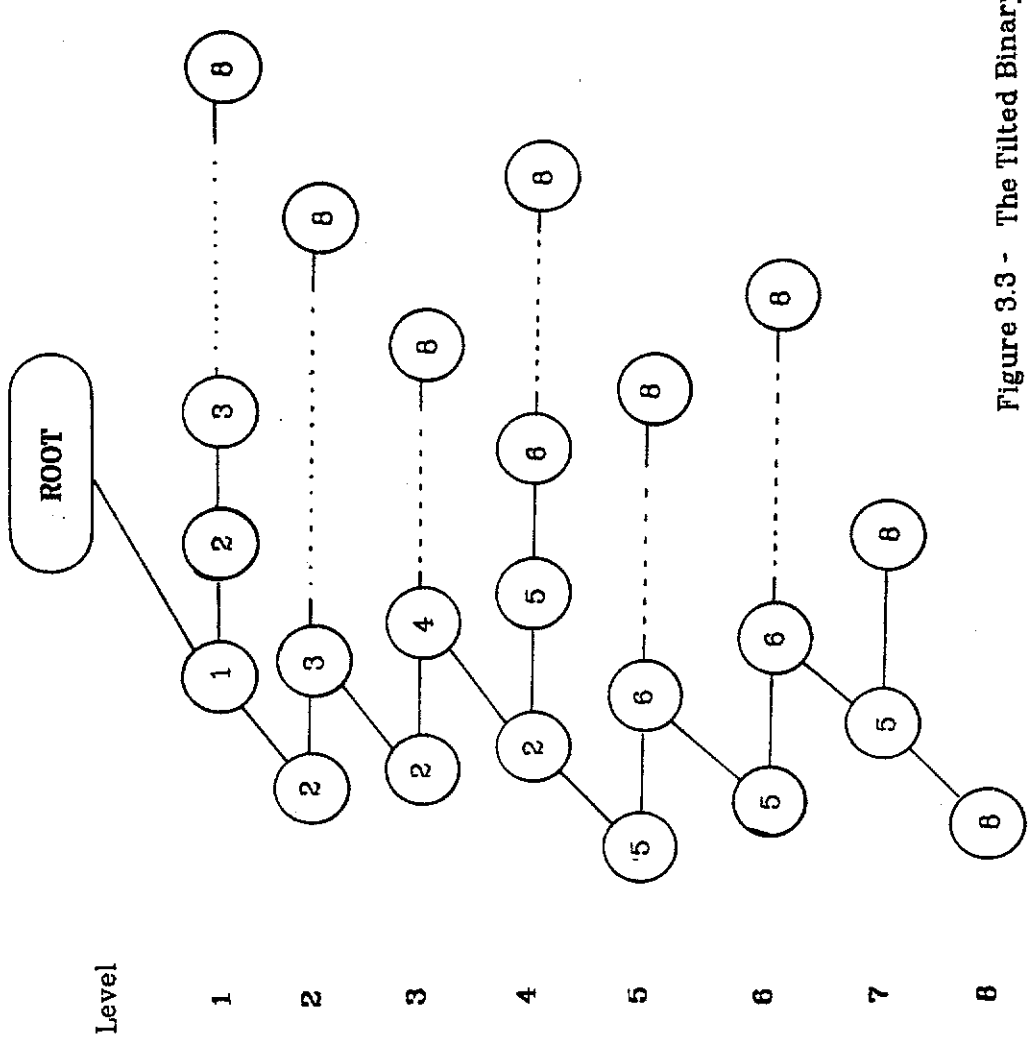


Figure 3.3 - The Tilted Binary Tree Representation.

using any order we need. (Preorder, Postorder or Inorder) It must be remembered that the level number of the tree corresponds to the corresponding input port.

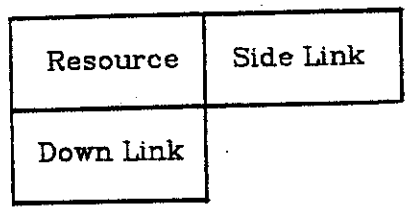
3.1.2.1.3. Search Algorithm: For each individual node, a simple structure of data, side pointer and down pointer are needed. The side link is used to link to the sibling on the right and down is used to link to the sub-tile one level below. (see Figure 3.4).

As for searching the tree itself, 3 stacks of SIDE NODE, its LEVEL and number of CONNECTIONS made so far, are needed. It also requires 3 lists of AVAILABLE OUTPUT PORTS, TREE LEVEL ARRANGEMENT, and REQUESTING INPUT PORTS.

In general, searching starts at one level down. The algorithm examines the node against the list of available output ports, then before going down any further, the next node to the right is put on the stack for future return when it finishes searching the left branch. Also its side node level and the number of connections succeeded so far is needed to be put on the stack. The list of tree levels arrangement is compared against the list of requesting input ports to decide which level of the tree needed to be searched. When it reaches the end of the left branch, it just pops the top of side node's stack and then starts searching the right branches. This whole procedure is terminated when the number of connections made is equal to the number of requesting input ports. It is also terminated if the whole tree is searched.

The worst-case searching is as follows: 40sec. to create a tree of approximately 17000 nodes; approximately 30 sec. to visit all the 17000 nodes for each;

A Node



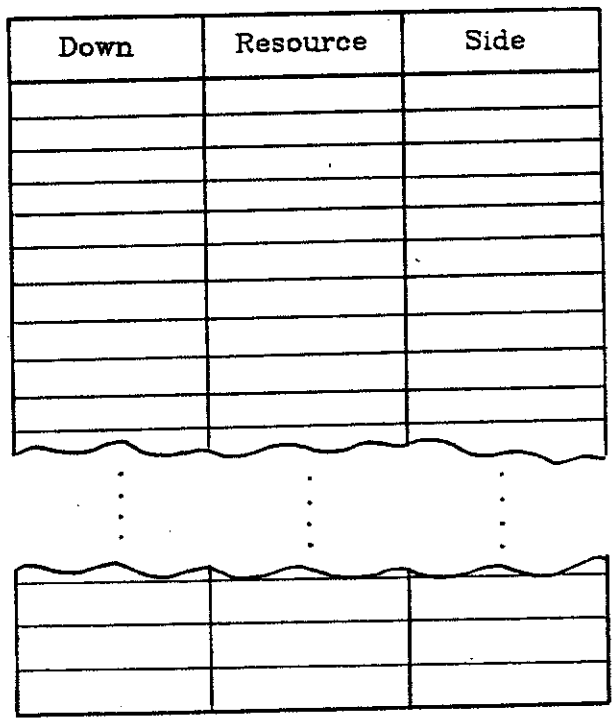


Figure 3.4 - Data Structure of the Tilted Binary Tree

worst-case search and T_{wc}=(65025*30) + 40 seconds which is around 542 CPU hours to perform 65025 searches

Because of the horrendous amount of search time required, a more time efficient method is needed.

3.1.2.1.4. Improved Search Algorithm: It is also possible to organize the binary tree with any order of levels rather than the original above. For instance, instead of having a tree organized with levels in monotonic order it can be organized as 23514678. There are $C\binom{8}{8}+C\binom{8}{7}+.....+C\binom{8}{1}$ ways of organizing the levels with result in 255 trees. The memory required may be too large. we decide to select $C\binom{8}{4}$ of the levels to result in 70 different trees.

And for the $C\binom{8}{8}+C\binom{8}{7}+.....+C\binom{8}{1}$ =255 ways that input ports can request connections, they are pretty well covered by the 70 trees and it takes less time to search the top of the tree than the other parts of the tree.

The search time for this method is much more efficient and is illustrated as followed: Since it takes approximately 30 sec. to visit about 17000 nodes, so it takes about .0018 sec. to visit one node. We need to create 70 files which takes a total of 2800 sec. Most of the time, the worst case search will be at the top 4 levels of the tree. Therefore, the search time how can be computed as:

T_{wc} = (65025 * 8 * 7 * 6 * 5 * 0.0018) + 2800 or approximately 56 cpu hours

1^{st} 2^{nd} 3^{rd} 4^{th} level

Note that this is the worst case time and it is much better than the previous one. In reality, it took about 10 hours or less to examine all 65025 cases.

3.1.2.1.5. Overhead of algorithm: For each binary tree created, it requires approximately 350K bytes file space to store it. There are also $\simeq$ 16.5K nodes of a bytes each. The creating time is $\simeq$ 38 sec. So it probably takes 30 sec. to search it for the worst case. In general, there are 70 or more of these binary trees which means that it requires approximately 24.5 megabytes of file space and 2.3 megabytes of core memory to be able to run the algorithm efficiently. And it takes $\simeq$ 30 sec. in the worst case to search for any particular connection. These requirements are tremendous for this case of 8x8 Binary n-Cube or Omega network. It certainly is not feasible for the case 16x16 or more. This calls for a more efficient algorithm that probably has a higher blocking probability but less computer space and time.

3.1.2.2. Centralized Scheduling Heuristics

As a comparison, we present a centralized heuristic and compare its performance against the optimal algorithm. Let

$$P_R = \text{Set of requesting processors} = \{P_i, P_{ii}, P_{iii}, \ldots P_x\}$$

$$R_A = \text{Set of free resources} = \{R_i, R_{ii}, R_{iii}, \ldots R_y\}$$

We assume that the processors and resources in P_R and R_A are ordered by their index numbers. A parameter of the heuristic is the number of retries ($0 \leq \text{RETRY} \leq y-1$). Supposed P_i fails to be connected to R_j due to a blocked connection, then the heuristic successively retries the next set of RETRY free resources to see if a connection can be made. Whether a

connection can be made within a fixed number of retries or not, the next processor in P_R is always matched with the first free resource in R_A immediately following the resource matched for the current processor. The heuristic, written in pidgin Algol, is shown in Figure 3.5.

To illustrate the heuristic, consider an 8 by 8 Omega network with $P_R = \{0,3,4,5\}$, $R_A = \{0,1,4,5\}$ (see Figure 3.9),

1. The algorithm connects P_0 to R_0 and is successful.
2. The algorithm connects P_3 to R_1 and is successful.
3. The algorithm tries to connect P_4 to R_3, but is blocked.

   If RETRY=0, then the algorithm connects P_5 to R_4 and is successful.

   If RETRY=1, then the algorithm tries to connect P_4 to R_4 and is successful. It continues to connect P_5 to R_3 and is successful.

For this example, the resource utilization is 100% if RETRY $\geq$ 1, otherwise, it is 75% for RETRY=0.

PROCEDURE Cent_heuristic

```
/* Assume that match (P_i, R_j) is a boolean procedure which
   returns TRUE if P_i can be connected to R_j and FALSE
   otherwise. If TRUE is returned, the connection is actually made.

   i - index of a requesting processor (i = φ means there is no
       requesting processor)

   j - index of a free resource (j = φ means there is no free resource)

   r - variable indicating the number of retries */

i=1; j=1;  /* initialization */

WHILE (i ≠ φ AND j ≠ φ) DO
   BEGIN
   done = FALSE; r=0;

   WHILE (NOT match (P_i, R_j) AND done .EQ. FALSE) DO
      BEGIN
      r = r+1;
      IF (r > RETRY)  /* Test for # of retries */
        THEN done = TRUE
        ELSE j ← next free resource in R_A;
      END;

   i ← next requesting processor in P_R;
   j ← next free resource in R_A;
   END

END
```

Figure 3.5 - Centralized heuristic for resource allocation.

3.2. Distributed Resource Scheduling Algorithms

3.2.1. Crossbar Network Design.

We present in this section the design of a Cross-bar switch to support distributed scheduling. The motivations behind studying Cross-bar switches are that it is non-blocking and it is very suitable for VLSI implementation. It has been shown that Cross-bar communication networks are favorable as compared to Banyan networks for VLSI implementation provided that the whole network is implemented on one chip [FRA80].

Figure 3.6 shows the overall structure of a Cross-bar network. Processor i, $0 \leq i < n$, initiates a request by sending a request signal to the switch along the i^{th} row. Resource j, $0 \leq j < m$, indicates that it is free by sending a resource signal along the j^{th} column. At cell $C_{i,j}$ where there are request and resource signals, the switch is set on and data transfer can begin. The request signal is removed from any further cells along the i^{th} row. Similarly, the resource signal is removed from any further cells along the j^{th} column. Each cell in the switch has enough intelligence to resolve the conflicts and to route the requests. There is a control latch in each cell to indicate the state of the switch. It is obvious that there is no centralized control for the routing of requests.

Because requests can appear and disappear at any time, it is important that a change in request state for one processor does not affect the state of allocation of other processors. To illustrate this, referring to Figure 3.6, if the request signal to cell $C_{i,j}$ is removed, then the latch in $C_{i,j}$ is reset and the

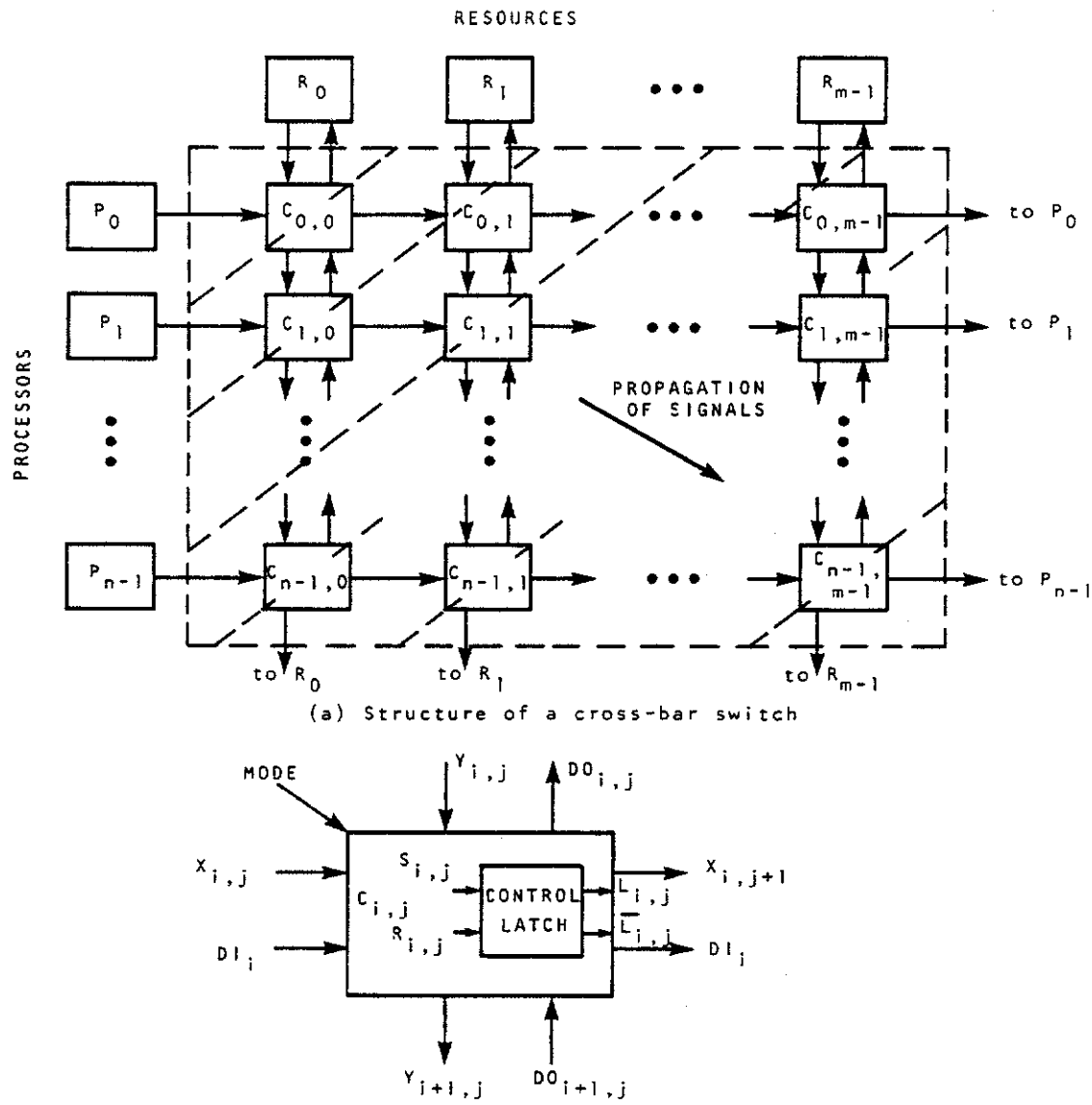


Figure 3.6 - A Cross-bar Switch to support Decentralized Scheduling

resource becomes free. The resource signal will again propagate down the j^{th} column. Processor k may have made a request previously. Since resource j was busy, it tried to search for another resource and found one. The new resource signal passed along the j^{th} column should be ignored in cell $C_{k,j}$ in order not to upset the state of a previous allocation. To solve this problem, we assume that the system operates in two modes: request mode and reset mode. In the request mode, processors can make requests for free resources. In the reset mode, processors can relinquish previously acquired resources. This method degrades performance because requests and resets cannot operate concurrently. However, a single signal line suffices to indicate which mode is active. Other alternatives which allow concurrency in requests and resets include (a) the use of state saving latches in each cell and (b) the use of separate request and reset control lines. These alternatives require more hardware and will be investigated in the distributed Omega and Binary n-Cube networks.

Referring to Figure 3.6(b), the inputs and outputs of cell $C_{i,j}$ which connects processor i and resource j have the following meaning:

$$X_{i,j} = \begin{cases} 0 & \text{processor i is not searching for a free resource} \\ 1 & \text{processor i is searching for a free resource} \end{cases}$$

(request mode)

$$X_{i,j} = \begin{cases} 0 & \text{processor i does not want change the state of allocation} \\ 1 & \text{processor i wishes relinquish the allocated resource} \end{cases}$$

(reset

mode)

$X_{i,j}$ always returns to 0 at the end of each mode

$$Y_{i,j} = \begin{cases} 0 & \text{resource j is busy and cannot accept any request} \\ 1 & \text{resource j is free and can accept a new request} \end{cases}$$

DI_i - data line to send data from the i^{th} processor

$DO_{i,j}$ - data line for the j^{th} resource to receive data from the i^{th} processor

$$L_{i,j} = \begin{cases} 0 & \text{Switch is off; any request made by processor i is passed the next cell} \\ 1 & \text{Switch is on; processor i is connected resource j} \end{cases}$$

MODE - controls the cell to be in request or reset mode.

The input/output relationship of the control signals is shown in the truth table in Table 3.1.

In the request mode, the latch is set ($S_{i,j} = 1$) if processor i is making a request and resource j is available. If resource j is not available ($Y_{i,j} = 0$), then the request signal is passed to the next cell ($X_{i,j+1} = X_{i,j}$). The resource signal to the next cell ($Y_{i+1,j}$) depends on the state of the control latch in the cell. If $Y_{i,j} = 0$, then $Y_{i+1,j} = 0$. If $Y_{i,j} = 1$ and $X_{i,j} = 1$, then the control latch is set and $Y_{i+1,j} = 0$. Since the $X_{i,j}$ signal returns to 0 at the end of the request mode, but the $Y_{i,j}$ signal may still be kept at 1, so $Y_{i+1,j}$ equals the output of the control latch ($\overline{L}_{i,j}$) when $X_{i,j} = 0$ and $Y_{i,j} = 1$. For those processors which have made requests previously, the state of allocation is not disturbed in the current request mode and data transmission can continue. In the reset mode, if processor i issues a reset signal, all the control latches in row i of the switch are reset. The logic equations for the controls and outputs are also shown in Table 3.1. The design of cell $C_{i,j}$ is shown in Figure 3.7.

The boundary connections for the switch are as follows. Each $X_{i,m}$ signal is connected directly back to P_i. Similarly, each $Y_{n,j}$ signal is connected back to R_j. Suppose P_i makes a request by setting $X_{i,0} = 1$ and it receives at the end of the request cycle, $X_{i,m} = 1$, this means that the request is not satisfied and P_i should resubmit its request in the next request cycle. Likewise, resource R_j indicates that it is free by setting $Y_{0,j} = 1$. If at the end of the request cycle, $Y_{n,j} = 1$, this means that the resource is not allocated and R_j should

Table 3.1 - Truth table and control signals for cell $C_{i,j}$

Inputs		Outputs			
$X_{i,j}$	$Y_{i,j}$	$X_{i,j+1}$	$Y_{i+1,j}$	$S_{i,j}$	$R_{i,j}$
0	0	0	0	0	0
0	1	0	$\overline{L}_{i,j}$	0	0
1	0	1	0	0	0
1	1	0	0	1	0

$X_{i,j+1} = X_{i,j}\ \overline{Y}_{i,j}$
$Y_{i+1,j} = \overline{X}_{i,j}\ Y_{i,j}\ \overline{L}_{i,j}$
$S_{i,j} = X_{i,j}\ Y_{i,j}$
$R_{i,j} = 0$
$DO_{i,j} = L_{i,j}\ DI_i + DO_{i+1,j}$

(a) Request mode

Inputs		Outputs			
$X_{i,j}$	$Y_{i,j}$	$X_{i,j+1}$	$Y_{i+1,j}$	$S_{i,j}$	$R_{i,j}$
0	0	0	0	0	0
0	1	0	1	0	0
1	0	1	0	0	1
1	1	1	1	0	1

$X_{i,j+1} = X_{i,j}$
$Y_{i+1,j} = Y_{i,j}$
$S_{i,j} = 0$
$R_{i,j} = X_{i,j}$
$DO_{i,j} = L_{i,j}\ DI_i + DO_{i+1,j}$

(b) Reset mode

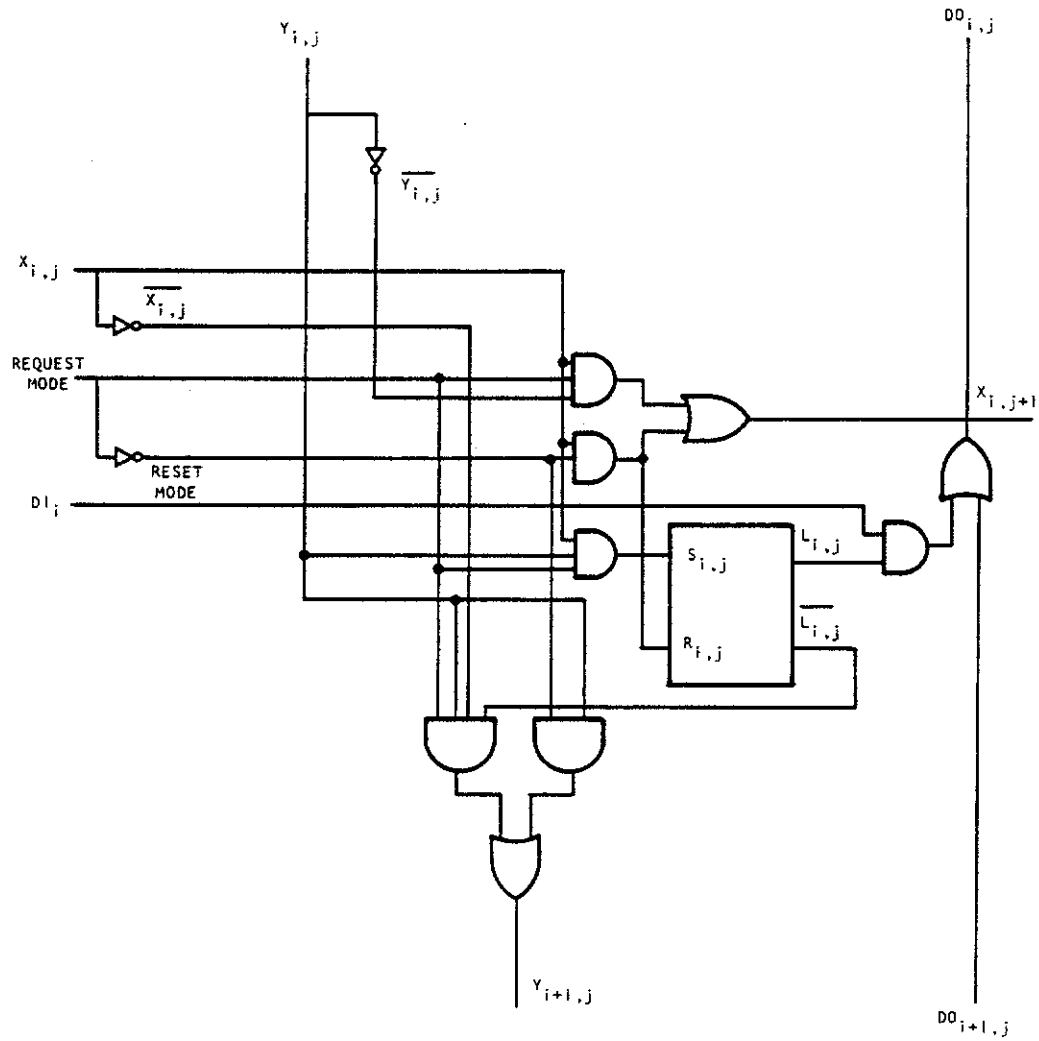


Figure 3.7 - A Cell in the Distributed Cross-bar switch.

send out the $Y_{0,j} = 1$ signal continuously. Otherwise, it will set $Y_{0,j} = 0$ to indicate that it is allocated.

Requests and resets are accepted at the beginning of the corresponding cycle. They are not accepted in the middle of a cycle because the next cycle cannot start until all the signals in the current cycle have settled. In each cycle, the signals propagate from the top left corner at 45° to the bottom right corner (Figure 3.6) in a wave-like motion. The maximum time for signal propagation is, therefore, proportional to n+ m. In the request cycle, the maximum gate delays in each cell is 4 because of two gate delays in the control latch. The maximum length of the request cycle is 4(n+ m) gate delays. In the reset cycle, the maximum delay in each cell is 1 due to the mode control gate. The maximum length of the reset cycle is (n+ m) gate delays.

A final remark about the scheduling algorithm is that it is asymmetric. That is, it favors processors with lower index numbers. In order to design an algorithm that is symmetric and to allow requests and resets to be initiated dynamically, more control lines are needed. Resources that are available can send a pulse of a short duration along a column. Only processors that receive a pulse will be assigned the resource. In this sense, the pulse behaves like a token. When different resources issue tokens randomly, the algorithm is symmetric.

3.2.2. Distributed Scheduling Algorithm for Networks with Logarithmic Delays.

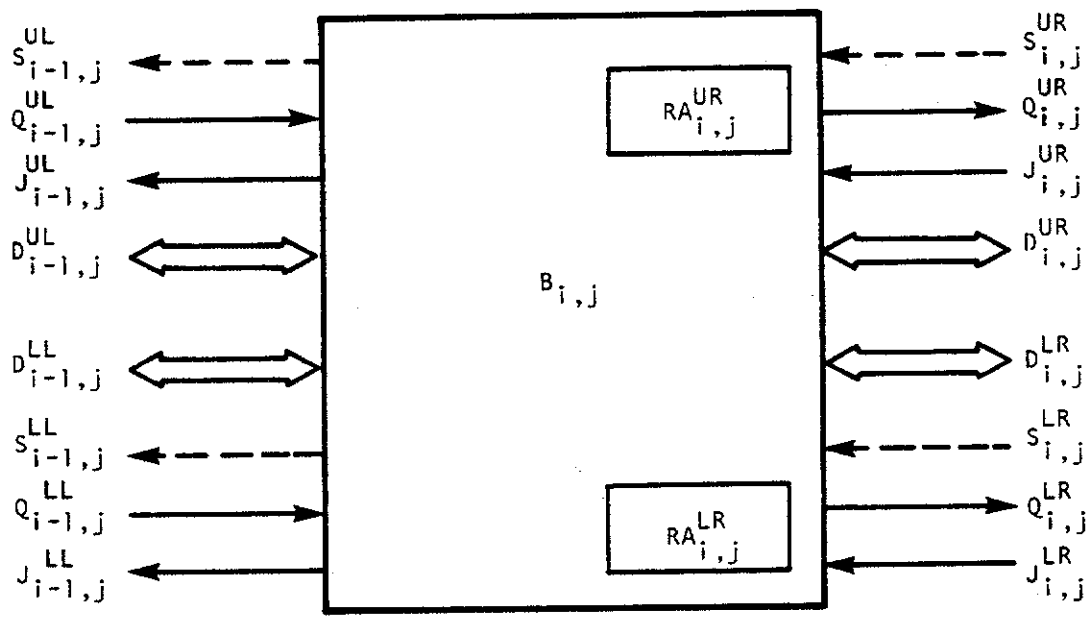
3.2.2.1.Simple case.

The centralized scheduling algorithm has a high overhead when the number of processors and resources to be scheduled is large since every requesting processor has to be scheduled sequentially. In a distributed algorithm, all the requesting processors are scheduled in parallel. The resource scheduling overhead is, therefore, proportional to the delay time in the network ($0(\log_2 N)$) and independent of the number of requesting processors.

The distributed algorithm is implemented by distributing the scheduling intelligence into the interconnection network so that there is no centralized control. Each exchange box can resolve conflicts and route requests to the appropriate destination. If a request is blocked, it will be sent back to the originating exchange box in the previous stage. Request routing is, thus, dynamic and all the exchange boxes operate independently.

Before the algorithm is described, some symbols must be defined. The information paths for exchange box j in stage i, $B_{i,j}$, is shown in Figure 3.8. There are four types of signals, S, Q, J and D:

S - carries information about the number of resources reachable from this link



information path in the resource phase

information path in the request phase

$B_{i,j}$-Exchange box j in stage i

data path

RA - resource availability register

Figure 3.8 - The information paths of an exchange box in the distributed scheduling algorithm.

$$Q = \begin{cases} 1 & \text{a request of a free resource is made on this link} \\ 0 & \text{otherwise} \end{cases}$$

$$J = \begin{cases} 1 & \text{a block has been detected in stages after the current stage and the request along this link is rejected} \\ 0 & \text{otherwise} \end{cases}$$

D - data transmission links

RA - resource availability register which stores the number of resources reachable from an output terminal of $B_{i,j}$.

For simplicity of representation, subscripts of symbols for signals incident upon and originating from $B_{i,j}$ are set to be i,j. The index of the box that they are connected to are not included in the representation as a mapping function. There are four types of superscripts, UL (upper left), UR (upper right), LL (lower left) and LR (lower right) and they indicate the corner of the exchange box that the signal links are connected to. The distributed scheduling algorithm utilize these signals to connect the data paths from the $(i-1)^{th}$ stage ($D^{UL}_{i-1,j}$, $D^{LL}_{i-1,j}$) to the $(i+1)^{th}$ stage ($D^{UR}_{i,j}$, $D^{LR}_{i,j}$).

Consider a situation when the network is completely free, and there is a set of requesting processors and free resources. All the resource availability registers are set to be zero initially. We will generalize later to the situation in which requests can be initiated dynamically.

The algorithm consists of two phases. In the first phase *(Resource Phase)*, information concerning the number of free resources is passed from the resource side to the processor side. Specifically, each resource that is free sends a "+ 1" along the S link to the exchange box connected to it. Referring to Figure 3.8, which shows exchange box $B_{i,j}$, the dashed lines represent the information flow in the resource phase. The exchange box receiving this information increments the corresponding resource availability registers. It then adds the two numbers stored in the two resource availability registers and sends the result to the two exchange boxes connected in stage i-1. Conceptually, the numbers $S_{i,j}^{UR}$ and $S_{i,j}^{LR}$ represent the number of resources reachable from the upper and lower output terminals of $B_{i,j}$. Therefore, the total number of resources reachable from this box is $S_{i,j}^{UR} + S_{i,j}^{LR}$ and this information is passed to the two exchange boxes connected in the previous stage along links $S_{i-1,j}^{UL}$ and $S_{i-1,j}^{LL}$. The delay for this phase is proportional to the number of stages of the network.

As an example, refer to Figure 3.9. Suppose processors P_0, P_3, P_4 and P_5 are making requests and resources R_0, R_1, R_4 and R_5 are free. Resource availability information is passed from the resource side to the processor side starting with stage 2. Box $B_{2,0}$ receives "+ 1" from R_0 and R_1. Therefore, it passes a "+ 2" to boxes $B_{1,0}$ and $B_{1,1}$. Likewise, box $B_{2,2}$ receives "+ 1" from R_4 and R_5 and passes this information to boxes $B_{1,2}$ and $B_{1,3}$. The propagation of this information is similar in stages 1 and 0. At the end of the resource phase, P_0, P_3, P_4 and P_5 knows that there are 4 resources available.

In the second phase *(Request Phase)*, the network propagates the requests from the processors to the resources. This uses the information that is obtained in the resource phase. The maximum total number of request and rejection signals pending in each exchange box is 2 since the exchange box can

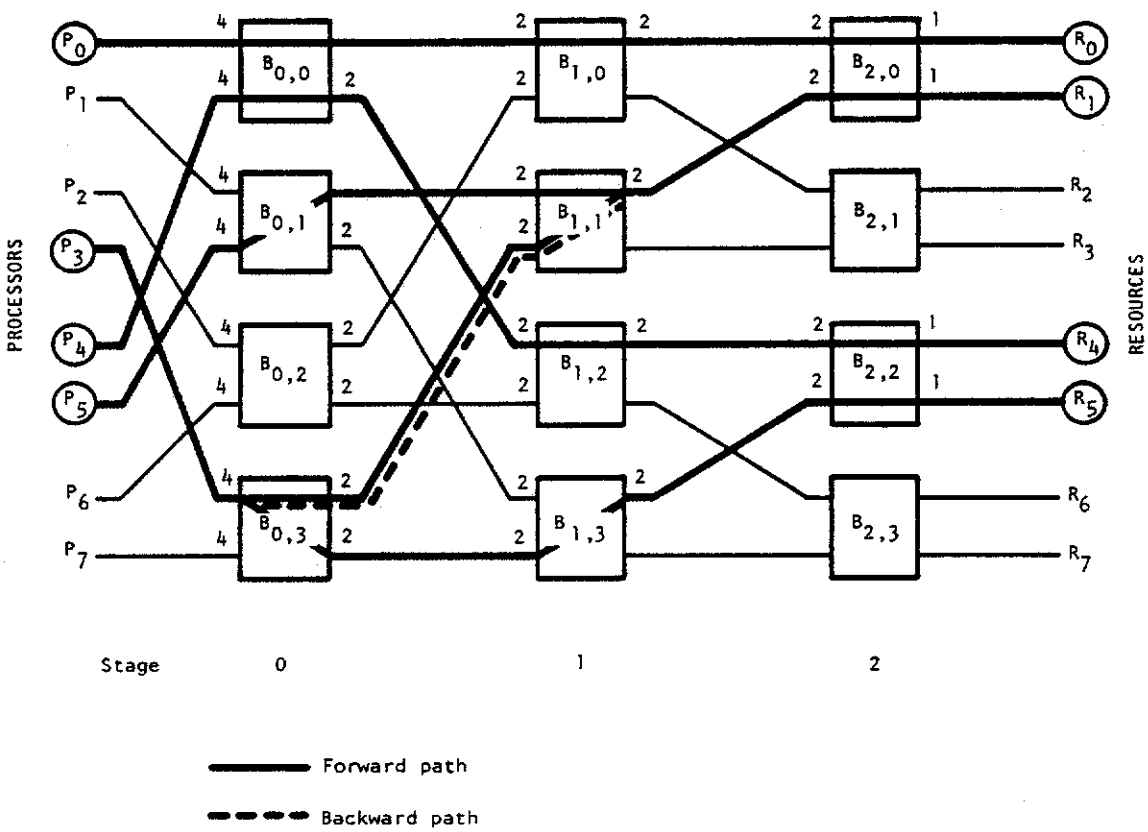


Figure 3.9 - Example of Omega network with 4 requesting processors and 4 free resources, (25% of requests are blocked and backtracked; 100% resource allocation; average delay = 3.5 units).

only make two connections at any time. For example, it is impossible to have two rejection signals received together with a request signal. Because in order for the rejection signals to be received, two request signals must have been received earlier. A new request cannot be received until the two previous requests have been rejected. Therefore, we can have any one of the following six combinations of signals pending in an exchange box: 2 Q's, 2 J's, 1 Q and 1 J, 1 J, 1 Q or no signal pending.

When multiple signals are pending in a box, priority must be set to determine the order of servicing these requests. Two priority rules are used:

(P1) For two request signals received ($Q^{UL}_{i-1,j}=1$, $Q^{LL}_{i-1,j}=1$), the request originating from the top input terminal ($Q^{UL}_{i-1,j}$) has priority over the other ($Q^{LL}_{i-1,j}$).

(P2) For one request and one reject signal received, the reject signal has priority over the request signal in service.

In servicing a request or a reject, two service rules are applied.

(S1) To service a request ($Q^{UL}_{i-1,j}$ or $Q^{LL}_{i-1,j}$), find a free output link where free resources can be accessed (contents of resource availability register is greater than zero). If both output links are free, then $S^{UR}_{i,j}$ is checked before $S^{LR}_{i,j}$. If such an output link is found, the output link is marked busy so that no further request can be made along this link and a request is sent to stage i+1. If the free output links do not lead to any free resources, a reject signal is sent from the original input terminal to stage i-1.

(S2) To service a reject ($J_{i,j}^{UR}$ or $J_{i,j}^{LR}$), the corresponding resource availability register ($RA_{i,j}^{UR}$ or $RA_{i,j}^{LR}$) is set to zero to indicate that no free resource is reachable from this output terminal. The output terminal is marked free and service rule (S1) is applied to search for another available output terminal where free resources can be reached.

For the six possible input combinations of signals pending in $B_{i,j}$, the sequence of priority and service rules applied is shown in Table 3.2.

If a request successfully reach a free resource, the resource sends a "-1" along the S link to the exchange box connected to it. For exchange box $B_{i,j}$ receiving a "-k" (k=1, 2,...) along the S link ($S_{i,j}^{UR}$=-k or $S_{i,j}^{LR}$=-k), if the content of the corresponding resource availability register is zero, then nothing is done. If not, the corresponding resource availability register is decremented and the "-k" information is passed to stage i-1 along $S_{i-1,j}^{UL}$ and $S_{i-1,j}^{LL}$. If both $S_{i,j}^{UR}$ and $S_{i,j}^{LR}$ are negative ($S_{i,j}^{UR}=-k_1$ and $S_{i,j}^{LR}=-k_2$), then both $RA_{i,j}^{UR}$ and $RA_{i,j}^{LR}$ are decremented and "$-(k_1+k_2)$" is sent along $S_{i-1,j}^{UL}$ and $S_{i-1,j}^{LL}$ to stage i-1.

Referring to the example in Figure 3.9, $B_{1,1}$ in stage 1 receives two requests. Since only one output terminal leads to free resources, the request originating from $B_{0,3}$ is rejected. This request, subsequently, finds another route via $B_{1,3}$ and $B_{2,2}$ to R_5. The average delay time is 3.5 units in this example (a unit is the time to pass through an exchange box)

The algorithm described above does not preclude dynamic operation. In fact, requests can be initiated at random times and they will be routed to a free resource or be rejected. The operation of the exchange box can be completely asynchronous. An accepted request is known to a processor when an acknowledgment is received along the data link. A request is rejected when a

Table 3.2 - Sequence of priority and service rules applied for the six possible combinations of signals pending in $B_{i,j}$

Combinations of signals pending in B(i, j)	Sequence of Priority and service rules applied
2 Q	P1, S1, S1
2 J	S2, S2
1 Q, 1 J	P2, S2, S1
1 Q	S1
1 J	S2
0 Q, 0 J	no action

rejection signal is received by the processor along the J signal link. A rejected request can be retried later.

3.2.2.2. Dynamic operation with Multiple-Resource Request.

To extend the problem into a dynamic environment, multiple resources are allowed in each request. The length of time that a resource is used may vary independently for each request. It is obvious that more resources are needed per output port. It is assumed that they are of the same type. A processor can make a request independently at any time, therefore, requests may arrive in certain distribution.

To facilitate the dynamic case, our exchange box is now of 4-function type which contains straight, diagonal, upper broadcast, and lower broadcast connections. The Binary n-Cube network is now a Generalized Cube with its broadcast capability [SIE81c]. The resources need to broadcast the latest information of their status to the network continuously. Therefore, requests generated by processors are now propagated through the network at the same time as broadcasting messages traveling in the opposite direction from the resources. A processor request can be classified as successful and unsuccessful . It is successful when all resources requested are allocated to the request in a given period of time. Otherwise it is an unsuccessful request. When a request is satisfied, it takes certain time for the processor to transmit the task to the resources. Once the transfer is completed the allocated resources can proceed to process the job and the requesting processor can relinquish the network to

others. The same is applied to an unsuccessful request. When a given period of time is up and all resources requested have not been allocated, the requesting processor also has to relinquish connections to minimize blocking and deadlock in the network.

In meeting a request requirement, the same algorithm described in the last section can be extended with a count on the number of resources requested by each request, and two additional registers $RR_{i,j}^{UL}$, $RR_{i,j}^{LL}$ are required in each exchange box to accommodate that count when it arrives. (RR–RequestRegister)

When a request enters an exchange box at either the top or bottom input port, using the information in $RA_{i,j}^{UR}$, $RA_{i,j}^{LR}$ & $RR_{i,j}^{UR}$ or $RR_{i,j}^{LL}$, the distributed algorithm will set the box and allocate the resources as follows:

- If $RR_{i,j}^{UL}$ ($RR_{i,j}^{LL}$) is less than any one or both of $\{RA_{i,j}^{UL}, RA_{i,j}^{LR}\}$, then the request is forwarded to the corresponding output terminal with less resources. For example, if $RR_{i,j}^{UL} < RA_{i,j}^{UR} < RA_{i,j}^{LR}$ then $Q_{i,j}^{UL}$ ($Q_{i,j}^{LL}$) is forwarded to the $UR_{i,j}$ output terminal.

- If $RR_{i,j}^{UL}$ ($RR_{i,j}^{LL}$) is greater than both but not the sum of $\{RA_{i,j}^{UR}, RA_{i,j}^{LR}\}$, then the request is broadcast to both output ports The RA that a large value is always chosen and the remainder is made up from the other RA. For instance, $Q_{i,j}^{LL}$ is making a request with $RR_{i,j}^{LL} = 9$. $RA_{i,j}^{UR} = 4$ and $RA_{i,j}^{LR} = 7$. $Q_{i,j}^{LL}$ is broadcast into 2 requests: the first one is forwarded to $LR_{i,j}$ output port with a count of 7 and the second request is assigned to the $UR_{i,j}$ output port with a count of 2.

- If $RR_{i,j}^{UL}$ ($RR_{i,j}^{LL}$) is greater than the sum of $\{RA_{i,j}^{UR}, RA_{i,j}^{LR}\}$ then the request is broadcast to the two output ports at their capacity and a reject signal is generated at the input port which carries a count of the difference of $RR_{i,j}^{UL}$ ($RR_{i,j}^{LL}$) and the sum of $\{RA_{i,j}^{UR}, RA_{i,j}^{LR}\}$. For instance, there is $Q_{i,j}^{UL}$ with $RR_{i,j}^{UL} = 14$ and $RA_{i,j}^{UR} = 6$, $RA_{i,j}^{LR} = 5$. Two requests are generated through $UR_{i,j}$ and $LR_{i,j}$ output ports with counts of 6 and 5 respectively. A reject $J_{i,j}^{UL}$ is forward to the previous stage of the network with a count of 3.

The 3 conditions mentioned above are also constrained by the availability of the output ports. In other words, the output port may have resources available but it may be busy serving another request. This condition of this output port is equivalent to $RA_{i,j} = 0$.

When there are 2 signals entering an exchange box, there will be no broadcasting involved in satisfying their requests because each request signal is entitled to one output port if both output ports are available. In allocating request for this case, whichever request is requesting more resources will have a priority over the other in the allocation to an output port which has more resources available. If its request is larger than an output port can satisfy, the difference of the request and $RA_{i,j}^{UR}$ ($RA_{i,j}^{LR}$) will be sent along the corresponding reject signal line. In other words, if a request signal at an exchange box are requesting more resources than each of its output ports can satisfy, there will be 2 requests forwarded through outputs ports at their capacity and 2 reject signals with counts on their differences respectively.

In addition, a reject signal is now being treated as a new incoming request signal in being serviced by an exchange box. For the dynamic case, we can

never have more than 3 signals at an exchange box simultaneously. There are a total of 7 possible cases of signals arriving at an exchange box: 2Q's, 2J's, 1Q & 1R, 1Q, 1J, 2J's & 1Q, and no signal.

3.2.2.3. Considerations for implementation.

In the simulation of a system which is controlled by the distributed algorithm, the static case only has 3 events: resource availability information from resources, requests from processors, and rejects from certain blocked exchange boxes. It is assumed that the delay through each stage is unity. This gives the blocking information pertaining to a specific network and the average delay time for requests to be allocated to available resources. The blocking information can be compared against the lower bound blocking probabilities of centralized algorithms.

In the dynamic case, there are 10 events involved:

1. Resources relinquish processing and become free.
2. Resources broadcast their availability.
3. Processors initiate requests.
4. Broadcast propagate through the network.
5. Requests propagate through the network.
6. Rejected signals are back-tracked.

7. Data is transferred from requesting processors to allocated resources.
8. Jobs are processed at resources.
9. Processors relinquish connections when data transfer is completed.
10. Processors relinquish connections when allocation is not met within a given period of time.

These events happen concurrently so it is essential and important to keep all the events synchronized in their time frame. Stacks are used to manipulate events in this system. As events occurring, statistics are collected [McD70].

Five main parameters to the system influence the performance of the network: the rate of jobs' arrival to the system, the length of time, the number of resources per output port of the network, a fixed period of time in which a request is allowed to search for resources and a duration of time during which data are transferred from the processors to the resources. These parameters were chosen for the meanings they serve. Data transfer time can determine how data amount and transmission rate may affect the network's performance. Jobs' arrival rate and jobs' processing time may be of variables with certain distribution which can assess the workload expected and the kind of computations that performed on the system. The allowable period of requests to search for free resources, may be used to tune the system's performance depending on other parameters. Lastly, number of resources per output port may be fixed depending on computational needs and resource availability.

The network starts out being completely free with all existing resources available to the network. Status messages from resources are broadcast to the processors through the network every unit of time. The inter-arrival time of requests is geometric. Each request may request 0 or more resources per request and not to exceed the maximum available resources on the network.

As status messages propagate from resources toward processors, requests from processors propagate through the network in the opposite direction. The requests will be directed through the network or be rejected by blocked exchange boxes. If a request is blocked and rejected from an interchange box, it will back-track to the previous stage and try to use another route. Each request is given a fixed time interval to search for free resources. If it finds all the resources required within the allowed period of time, it is a successful request. Otherwise, the request will be terminated. Once a request is successful, the task will be transmitted from the processor to the allocated resources (satisfying some distribution function). After the data transfer is completed, the network link is broken. The allocated resources start to process the given job for the requested length of time. When the job is finished, th resources are relinquished and new status messages are broadcast through the network.

The performance of the network is clearly affected by the workload arriving at the processors, the length of each job, data transfer time, and time allowed to search for free resources. This dynamic case is a more realistic representation of the operation. Performance results of the network will be discussed in the next chapter.

CHAPTER IV - RESULTS AND DISCUSSION

4.1. Centralized Scheduling Algorithm

Simulation results presented in [FRA80] shows that with N=8, there is a blocking probability of about 30% using address mapping. The results obtained by exhaustive enumeration over all possible combinations of connections for a subset of requesting processors and free resources using the Centralized Scheduling Algorithm show virtually no blocking when the Omega and the Binary n-Cube networks operate as RSIN's. This set of results is the lower bound blocking probability for the network and is regarded as a benchmark for comparison with other algorithms.

The Omega network is equivalent to the Binary n-Cube network with the difference that it operates in the reverse direction. Using these networks as RSIN's, they are also statistically identical. The performance of these networks is evaluated by selecting a random subset of processors and resources and finding the maximum number of resource allocations. If the Omega network can be re-arranged into a Binary n-Cube network, then their performance as RSIN's are identical. This rearrangement is exemplified in the Omega network in Figure 3.9. If $B_{0,1}$ and $B_{1,1}$ are moved so that they are adjacent to $B_{0,3}$ and $B_{1,3}$, and with proper re-labeling of processors and resources, the Omega

network is transformed into a Binary n-Cube network. Therefore, the performance of the Binary n-Cube and Omega networks are equivalent.

From the results plotted it is seen that the blocking probability and standard deviation are very small (Figure 4.1, 4.2). We can conclude that with a good scheduling algorithm, the Binary n-Cube or the Omega network serves almost equally as well the distributed Crossbar switch for resource sharing.

4.2. Centralized Scheduling Heuristics

The procedure *Cent_heuristic* in Figure 3.5 has a complexity of $O(\log_2 N)$. (It is proportional to the number of stages in the network). The worst case complexity of the heuristic for x requesting processors and y free resources ($0 < x,y \leq N$) is, therefore, $O(N(RETRY+1)\log_2 N)$. If RETRY=0, the heuristic has complexity $O(N*\log_2 N)$. If RETRY=N-1, the heuristic has complexity $O(N^2\log_2 N)$.

Since the heuristic assumes a predetermined sequence of allocations and no backtracking is provided if a wrong decision is made, the heuristic is suboptimal. The performance of the heuristic with RETRY=0 and RETRY=8 are shown in Figures 4.3 - 4.10. It is seen that the blocking probability is higher than the optimal case (around 7%) and its standard deviation is also higher. As the number of retries increases, the blocking probability reduces. In general, the standard deviation of the retry case always seems to be better.

The Omega and the Binary n-Cube networks have different performance on the centralized heuristic. This is due to the fact that the order in which

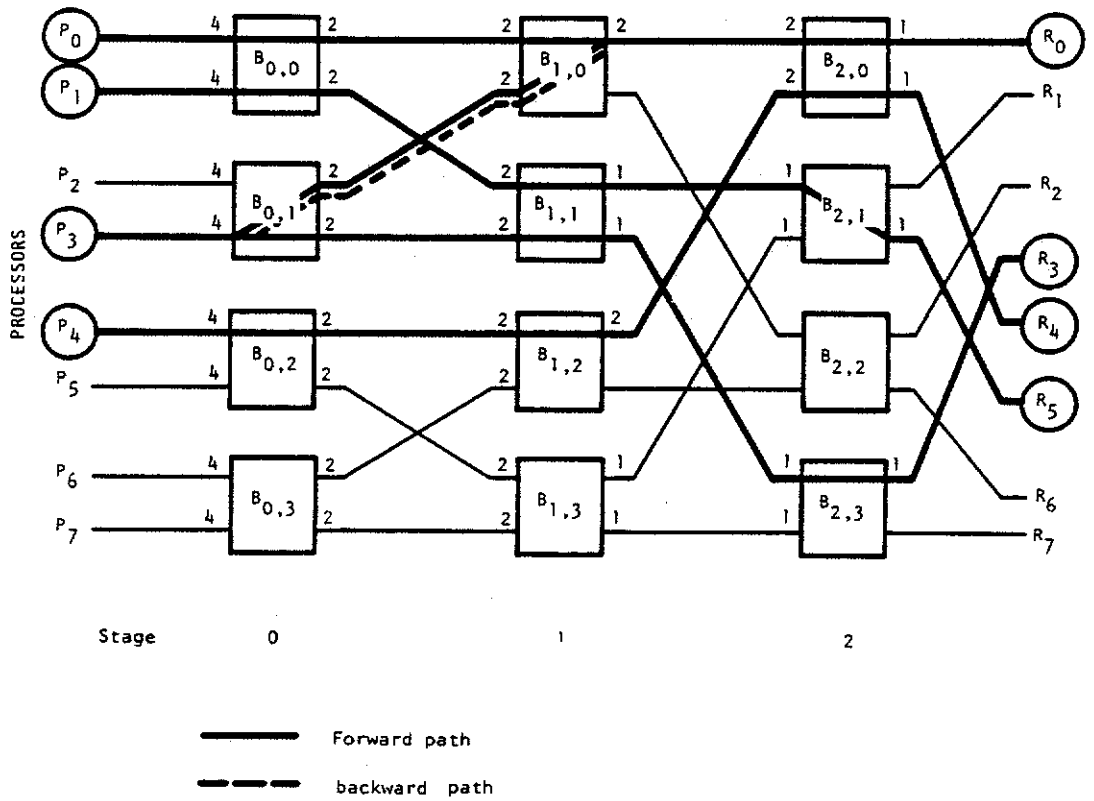


Figure 3.10 - Example of Cube network with 4 requesting processors and 4 free resources, (25% of requests are blocked and backtracked; 100% resource allocation; average delay = 3.5 units).

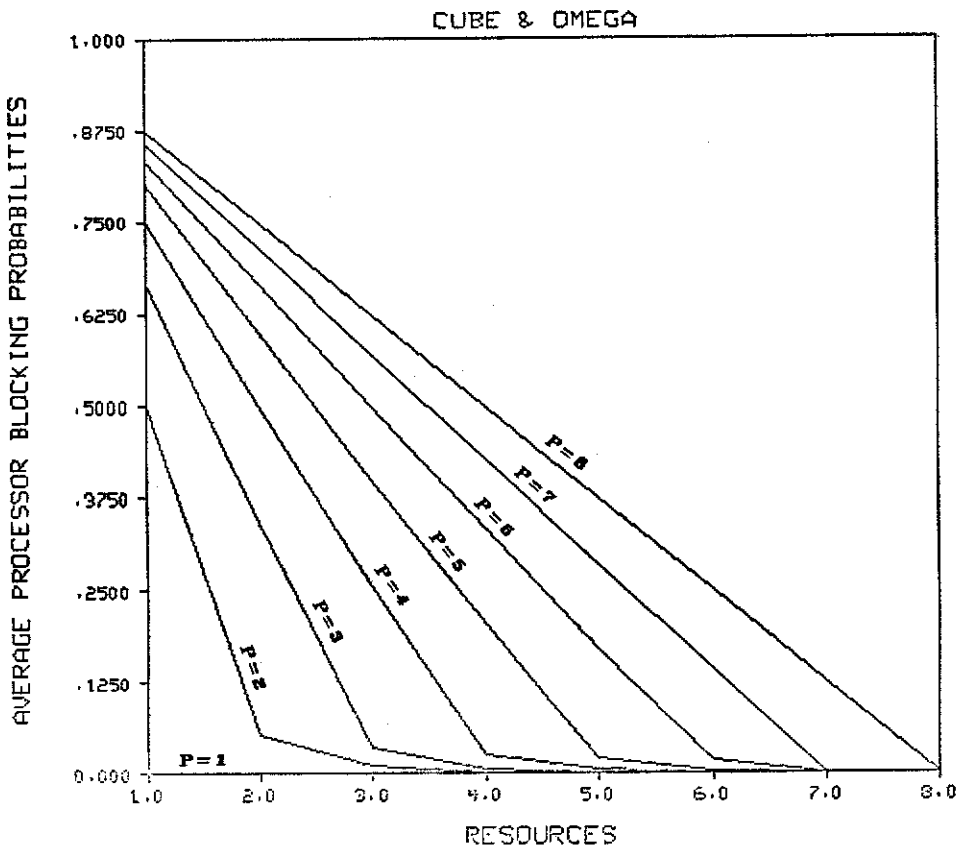


Figure 4.1 - Average Processor Blocking Probabilities for the 8x8 Cube and Omega Interconnection networks using Optimal Centralized Scheduling Algorithm.

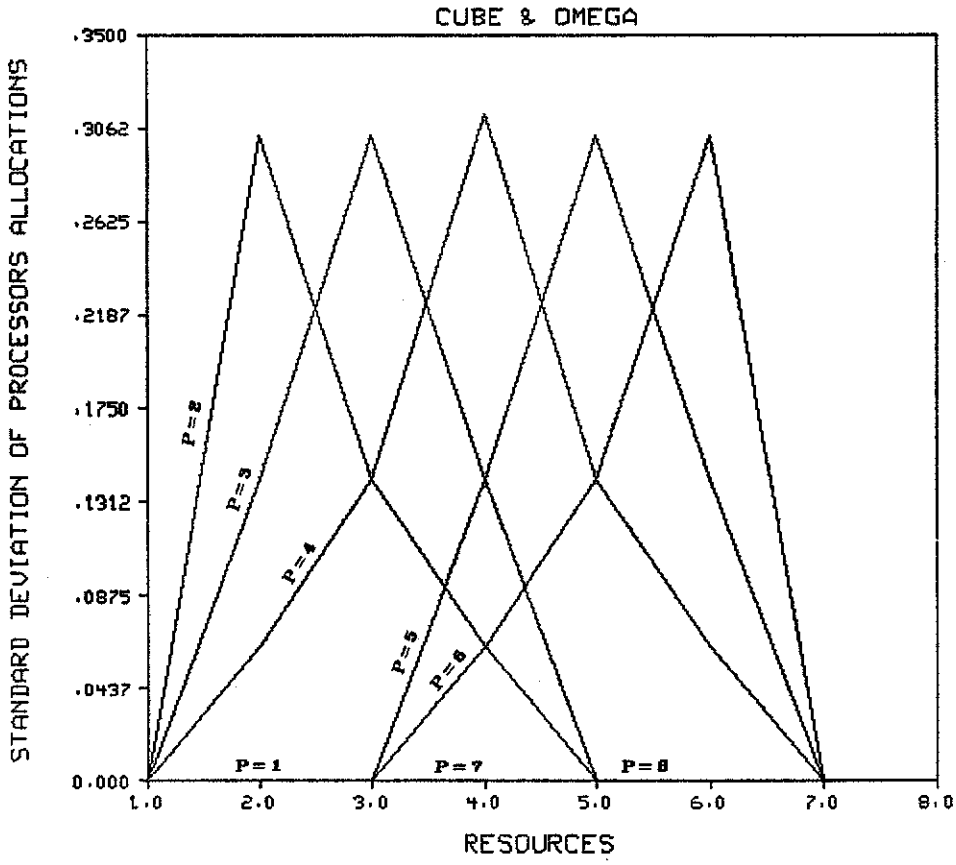


Figure 4.2 - Standard Deviations of processor Allocations for the 8x8 Cube & Omega Interconnection networks using Optimal Centralized Scheduling Algorithm.

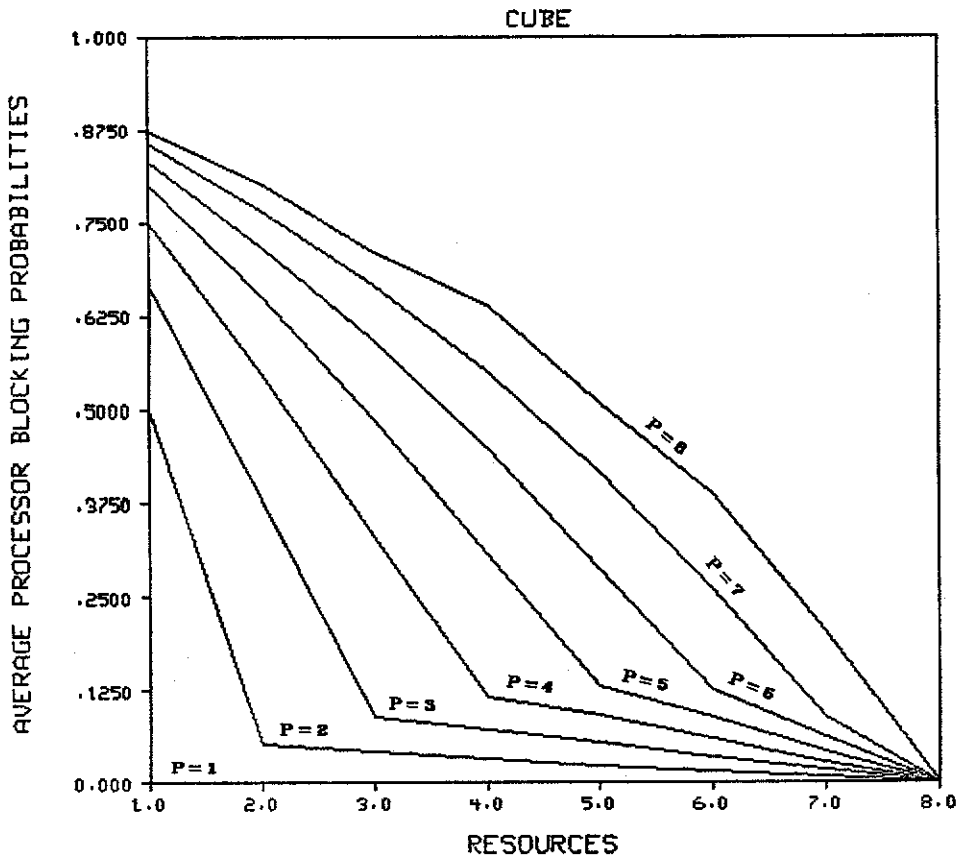


Figure 4.3 - Average processor Blocking Probabilities
for the 8x8 Cube Interconnection network
with no "Retry" Centralized control Heuristics

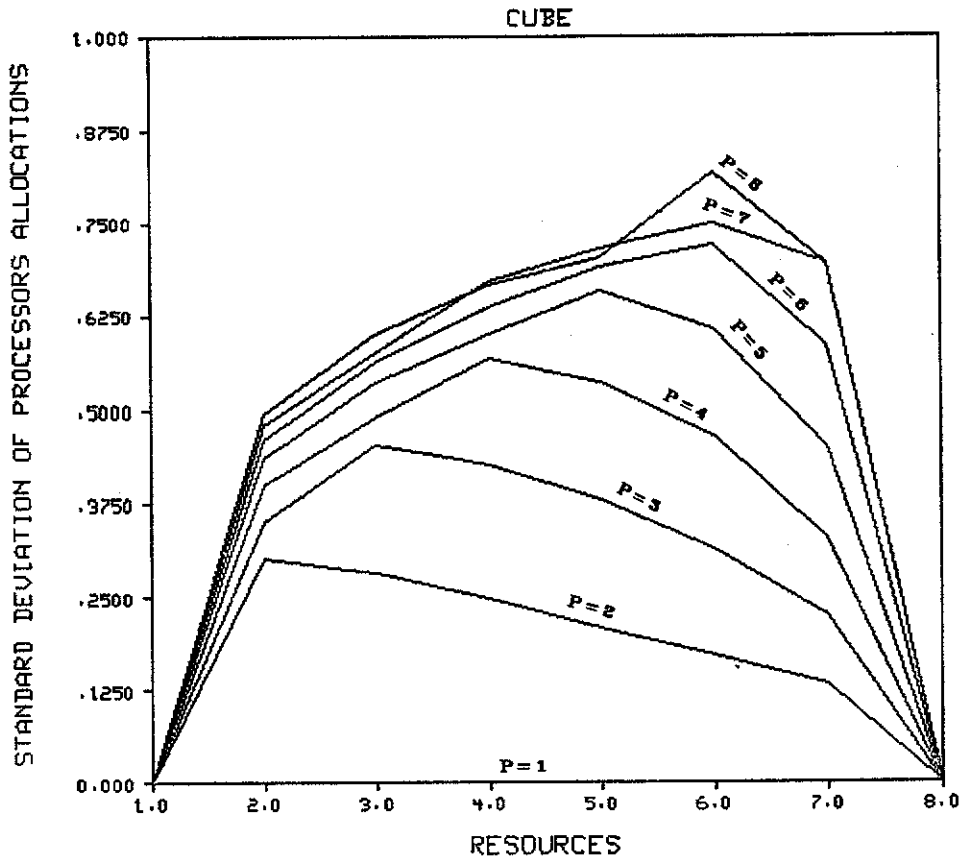


Figure 4.4 - Standard Deviations of processor Allocations for the 8x8 Cube Interconnection network using "No Retry" Centralized control Heuristics.

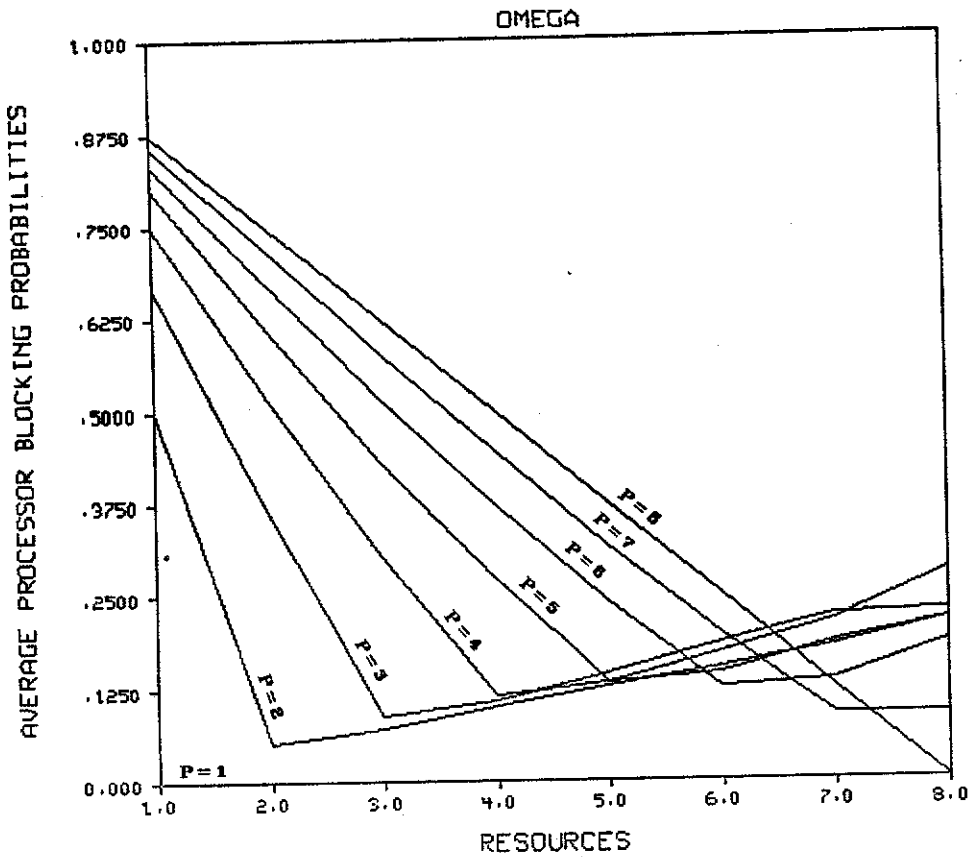


Figure 4.5 - Average processor Blocking Probabilities for the 8x8 Omega Interconnection network using "No Retry" Centralized control Heuristics

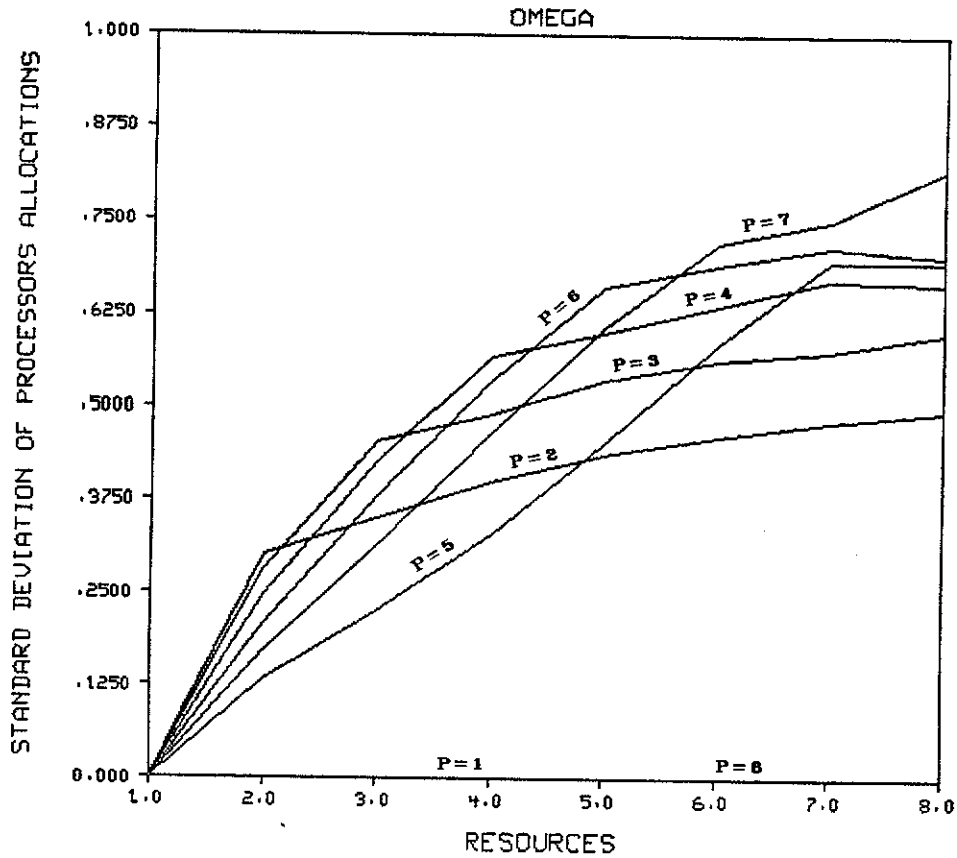


Figure 4.6 - Standard Deviations of processor Allocations for the 8x8 Omega Interconnection network using "No Retry" Centralized control Heuristics.

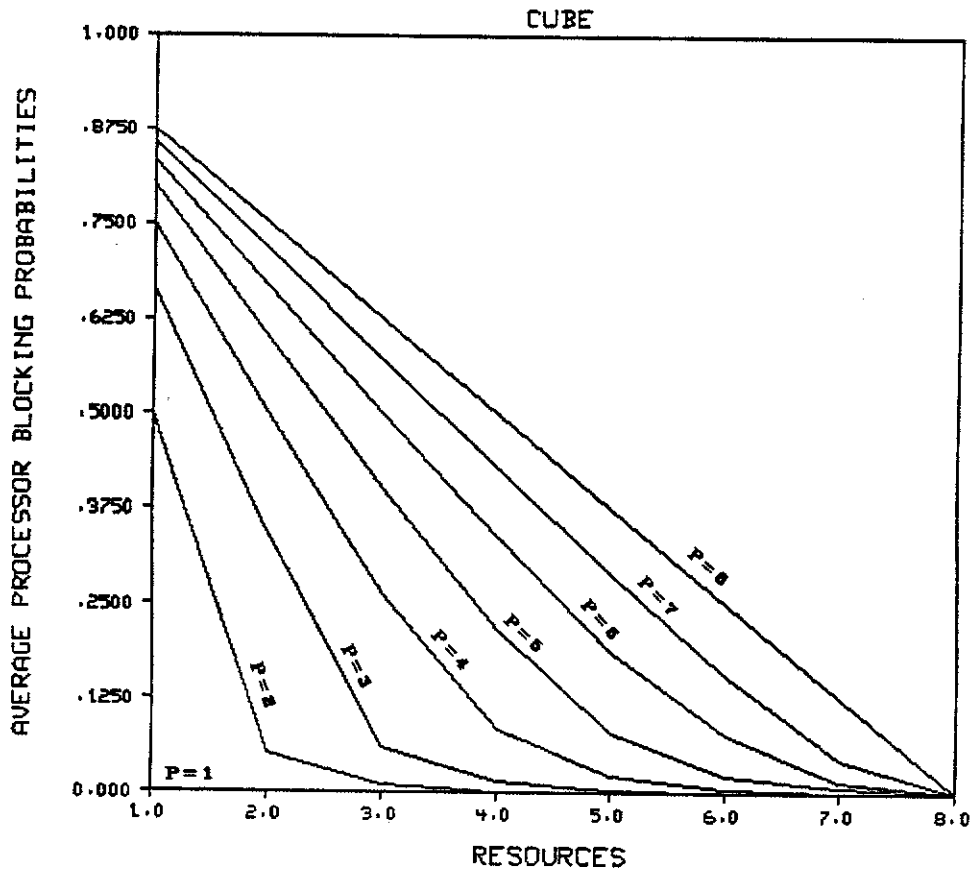


Figure 4.7 - Average processor blocking probabilities for the 8x8 Cube Interconnection network using "Retry" Centralized Control Heuristics (Retry = 8)

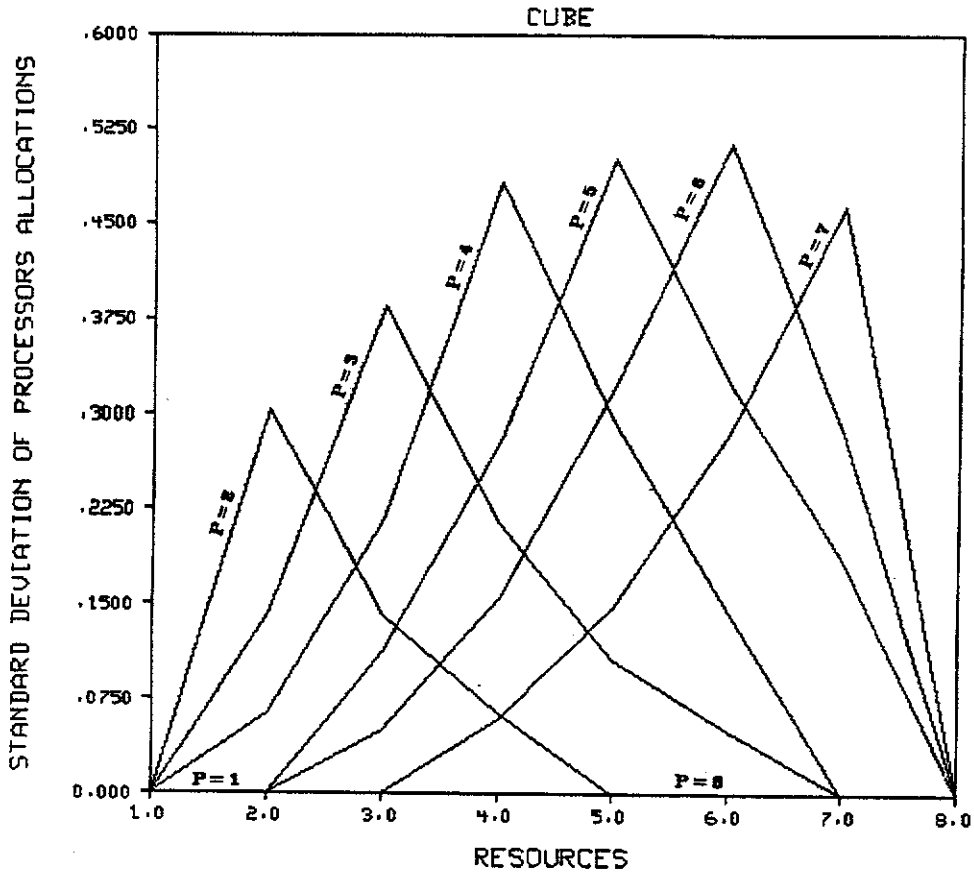


Figure 4.8 - Standard Deviations of processor Allocations for the 8x8 Cube Interconnection network using "Retry" Centralized control Heuristics.
(Retry = 8)

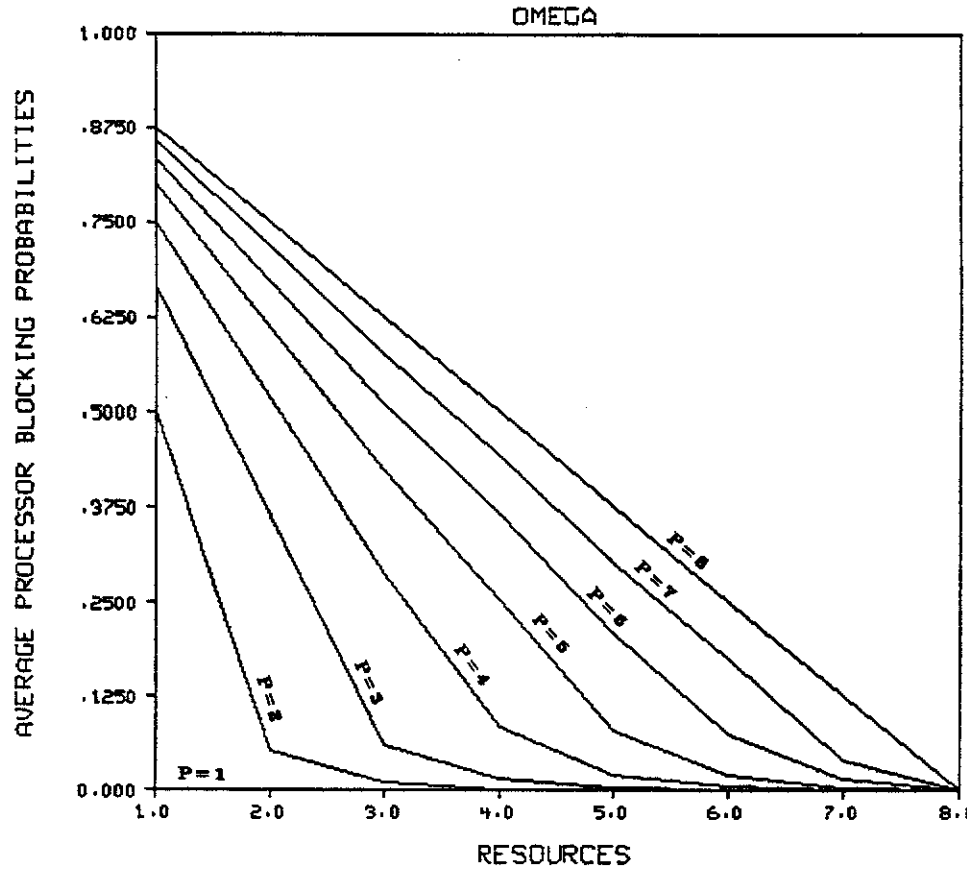


Figure 4.9 - Average processor blocking probabilities for the 8x8 Omega Interconnection network using "Retry" Centralized control Heuristics.
(Retry = 8)

resources are tried is different in the two networks. The No retry heuristics always tries to map processors to available resources one-to-one and do nothing if connecting effort fails. Because of the its straight forward manner of allocating resources, the Omega network incurred higher blocking when more resources are available. Finally, in the case of RETRY=0, the Omega and Binary n-Cube networks have identical performance when the number of processors and resources are equal. Elsewhere, the Omega network has worse performance.

4.3. Distributed Scheduling Algorithm

4.3.1. Static Case

Unlike the centralized scheduling algorithm's large overhead of sequential scheduling when the number of processors and resources is large, distributed scheduling algorithm schedules resources in parallel and has a delay time in scheduling proportional to $O(\log_2 N)$.

The performance of the distributed algorithm is again plotted in Figures 4.11 - 4.14 and it is identical for the Omega and Binary n-Cube networks. It is seen that the blocking probability is less than 20% in all cases and compare favorably with the optimal algorithm and centralized heuristic. The standard

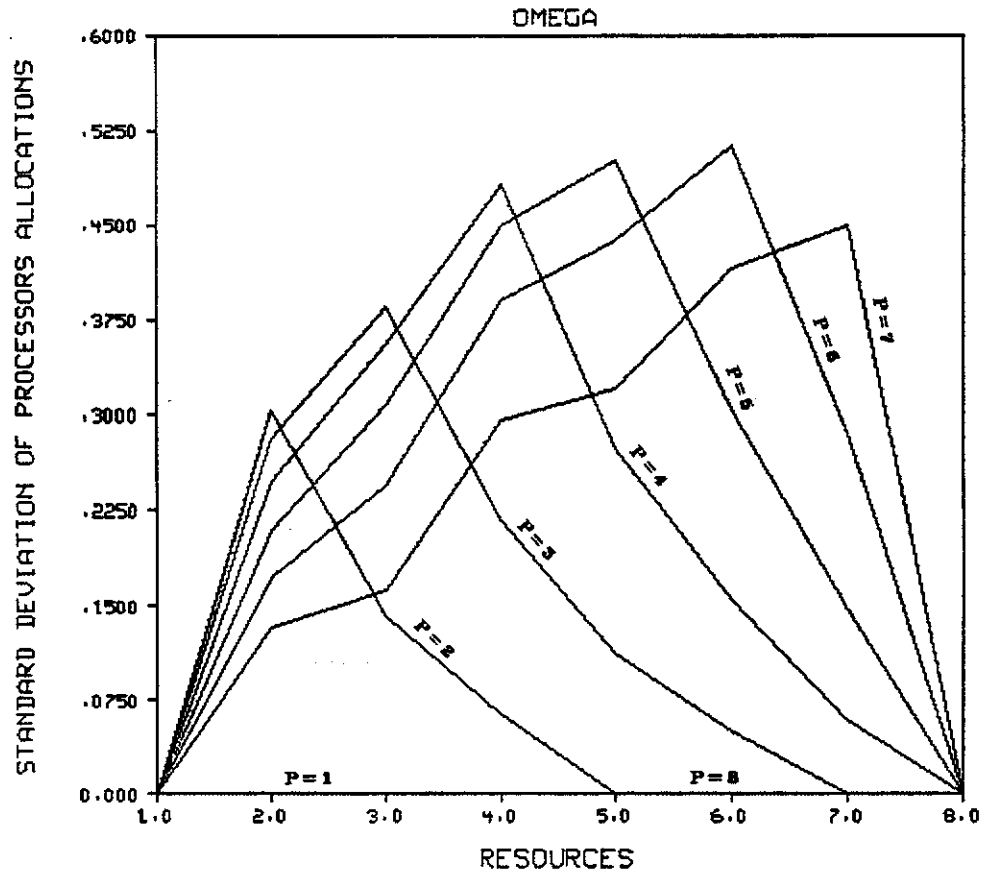


Figure 4.10 - Standard Deviations of processor Allocations for the 8x8 Omega Interconnection network using "Retry" Centralized control Heuristics.
(Retry = 8)

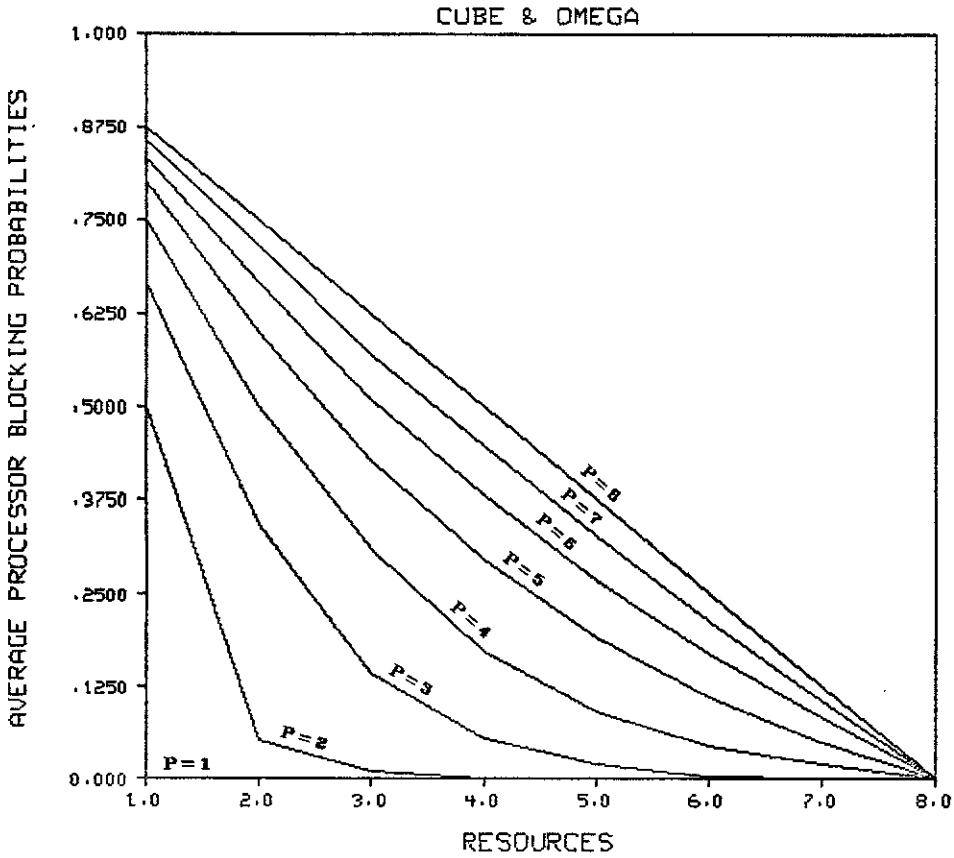


Figure 4.11 - Average processor Blocking Probabilities for the 8x8 Cube & Omega Interconnection network using Distributed control algorithm.

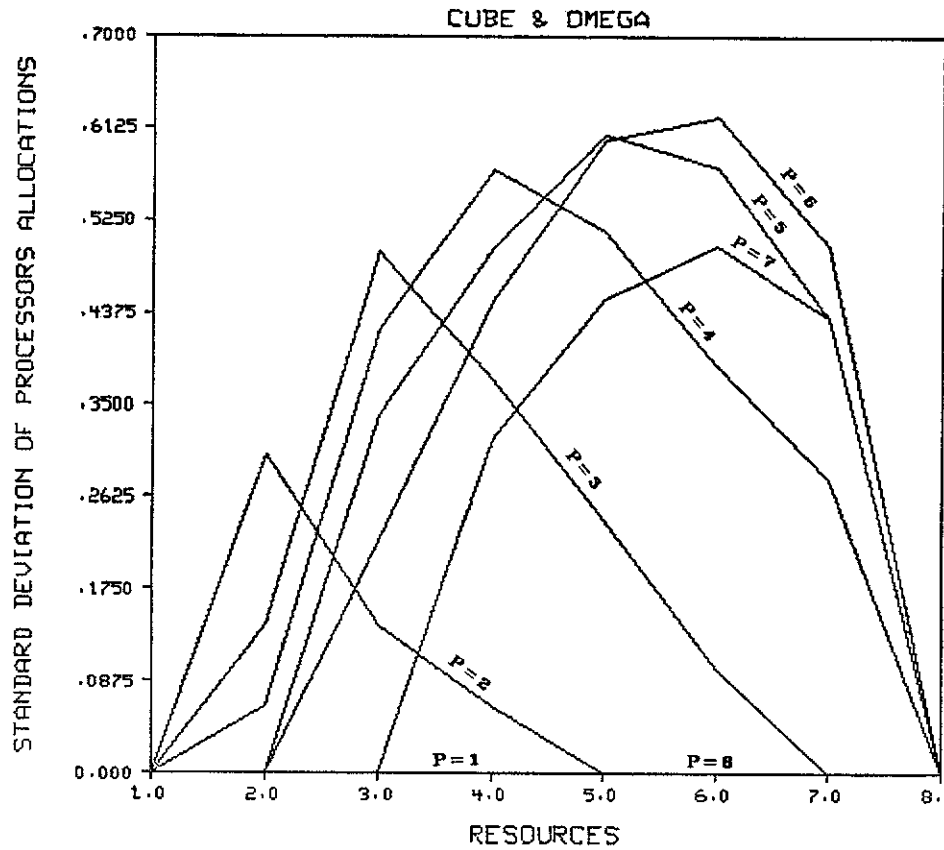


Figure 4.12 - Standard Deviations of processor Allocations for the 8x8 Cube & Omega Interconnection networks using Distributed control algorithm.

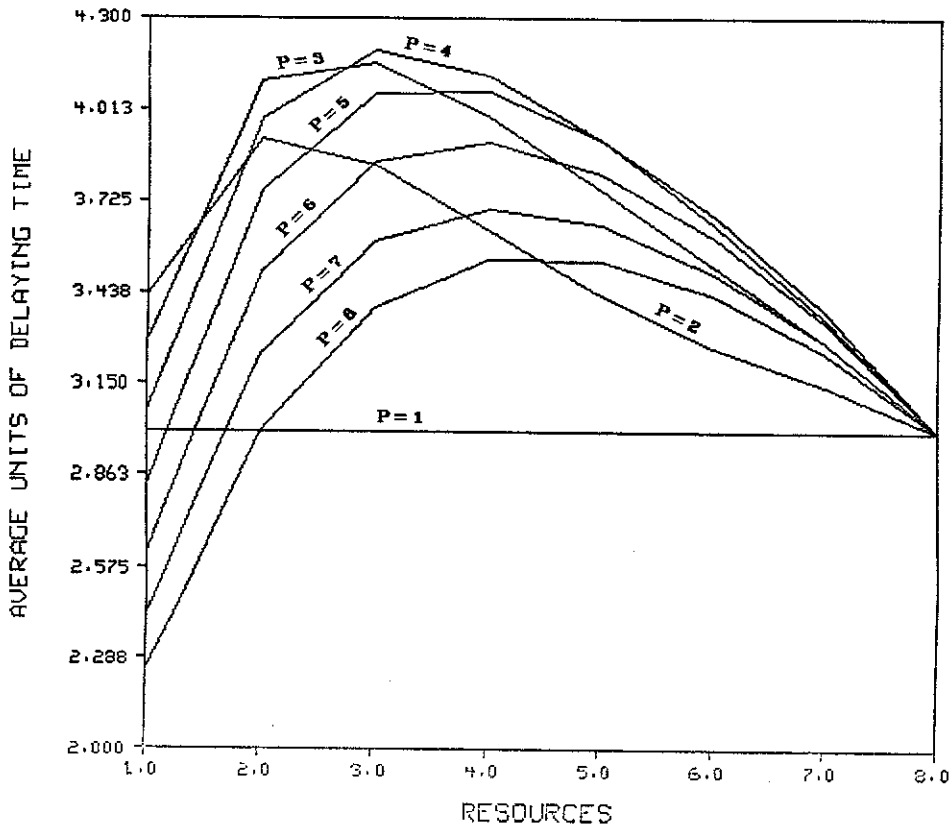


Figure 4.13 - Average Delay time for Distributed control algorithm in the 8x8 Cube and Omega networks. (A unit is the time it takes to pass through an exchange box).

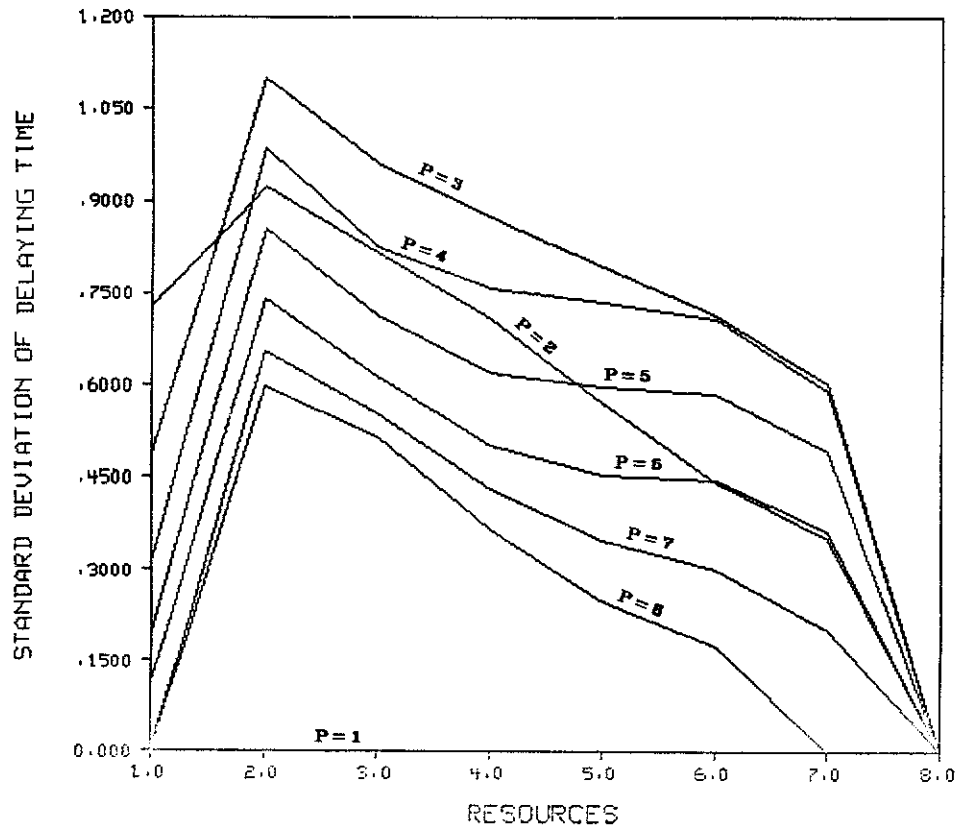


Figure 4.14 - Standard Deviations of Delay time for Distributed control algorithm in the 8x8 Cube and Omega network. (A unit is the time it takes to pass through an exchange box.)

deviation is approximately doubled as compared with the optimal case. The average delay time for a request to access a free resource or be rejected is shown in Figure 4.13. The delays is never greater than 4.2 units of time in which the delay through an exchange box is 1 unit. The delay time of the algorithm is dependent on the delay in the network and not on the number of requesting processors.

Figures 4.15 and 4.16 give 2 representative comparisons among the algorithms. These results are plotted out from the diagonals of output results matrices (Appendix). Therefore, the numbers of requesting processors are equal to the numbers of free resources.

4.3.2 Dynamic Case

For ease of illustrating the dynamic case using the distributed algorithm, a few acronyms are defined;

- RT: The average length of time a request will be processed by resources if successfully allocated.

- RWT: A period of time during which a request is allowed to search for free resources. Whenthis time is exhausted, the corresponding request has to relinquish the network and all partially allocated resources.

- DT: A length of time during which data will be transferred from processor to the allocated resources once the connection is made.

- P_R: Probability that a processor makes a request for i resources, i=0,1,2,...,n. (n is the total number of resources existing in the network.) In

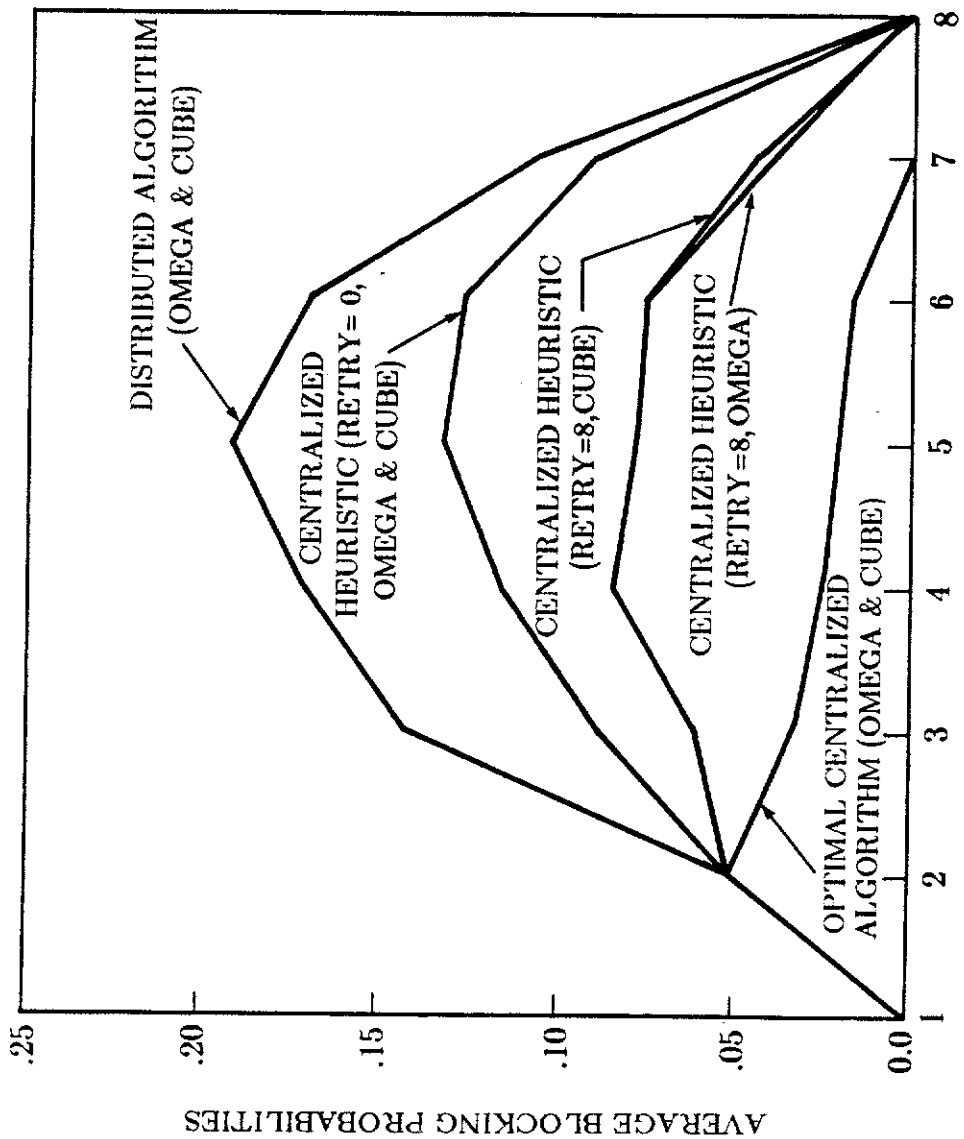


Figure 4.15 - Blocking Probability for Resource Allocations on Omega & Cube Networks (N=8).

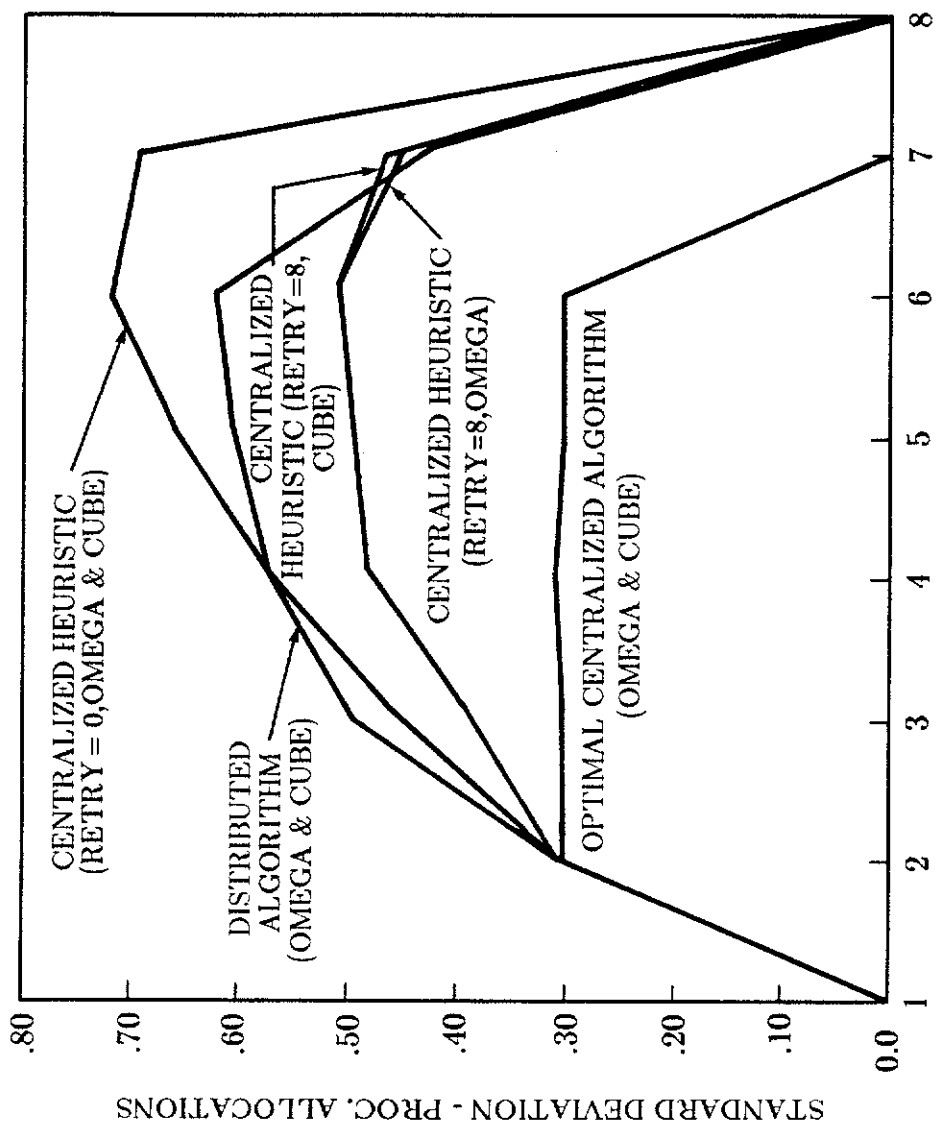


Figure 4.16 - Standard Deviation of Number of Requesting Processors allocated for Omega & Cube networks (N=8).

this case, a geometric distribution is used. The inter-arrival time of requests in the network can be computed using $\frac{1}{P_r(x=0)}$ {$P_r(x=0)$ is the probability of a processor not making a request.}

In general, average network delay time is increased at RWT increases. It is obvious that the longer a request is allowed to search for free resources, it will have a higher chance bouncing back and forth until the allowed time is up. This results in a higher delay time.

RT reflects the type of jobs in the system. The longer the jobs are, the less resources are available to the network. In general, longer jobs introduce higher blocking in the system. RT affects the system in the same way as RWT. In addition, it is noted that there is an optimum point where blocking is the lowest.

DT affects the system in the same way as RWT.

The work load in the system is as important as the type of jobs it processes. The lower the P_r the heavier the workload. P_r behaviors can be viewed in the same manner as that of RT.

In conclusion, there are tradeoffs in the system among RT, P_r, RWT and DT. Depending on the applications, it is found that the distributed algorithm serves well in controlling the traffic and the delay time is proportional to the number of stages in the network (Figure 4.17 - 4.24).

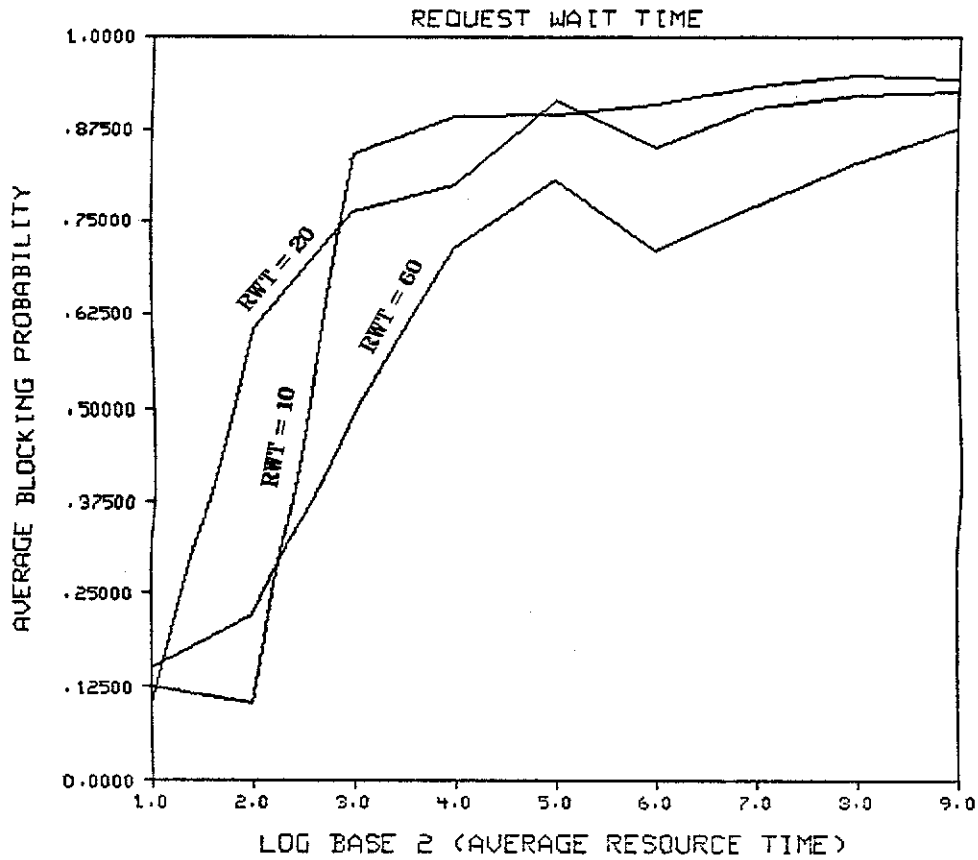


Figure 4.17 - Average Blocking Probability of the dynamic case of the distributed algorithm from the effects of Resource Time and Request Wait Time

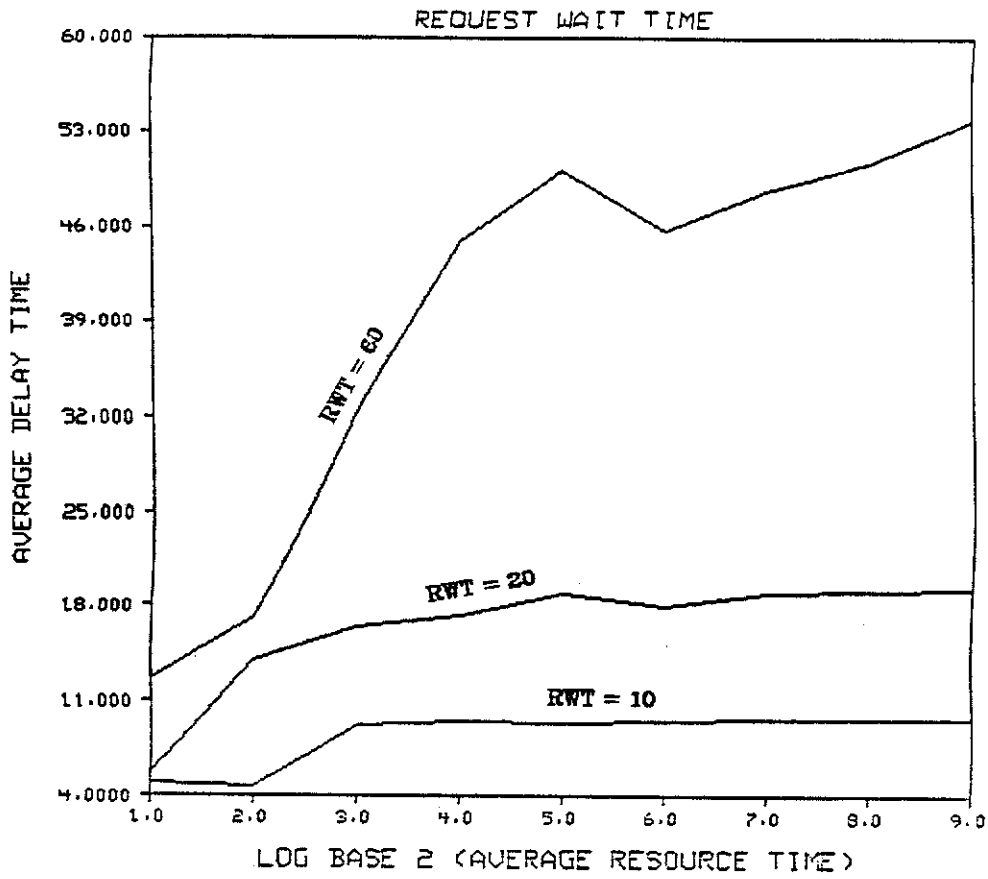


Figure 4.18 - Average Request Delay Time of the dynamic case of the distributed algorithm from the effects of Resource Time and Request Wait Time

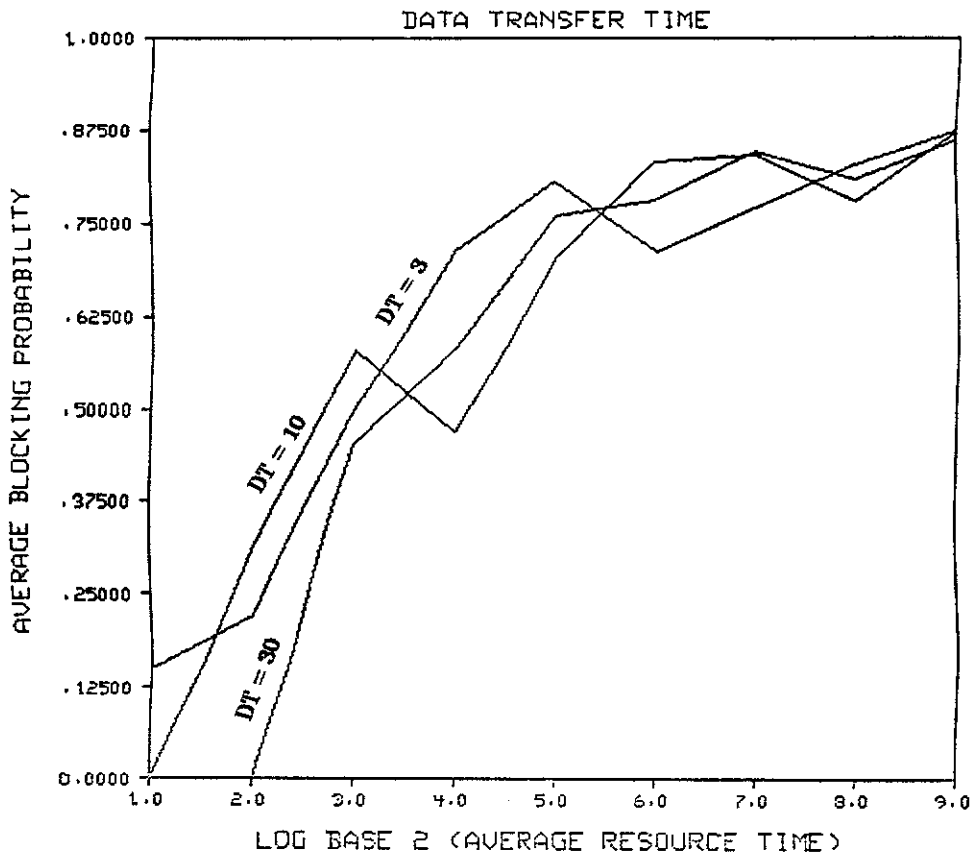


Figure 4.19 - Average Blocking Probability of the dynamic case of the distributed algorithm from the effects of Resource Time and Data Transfer Time

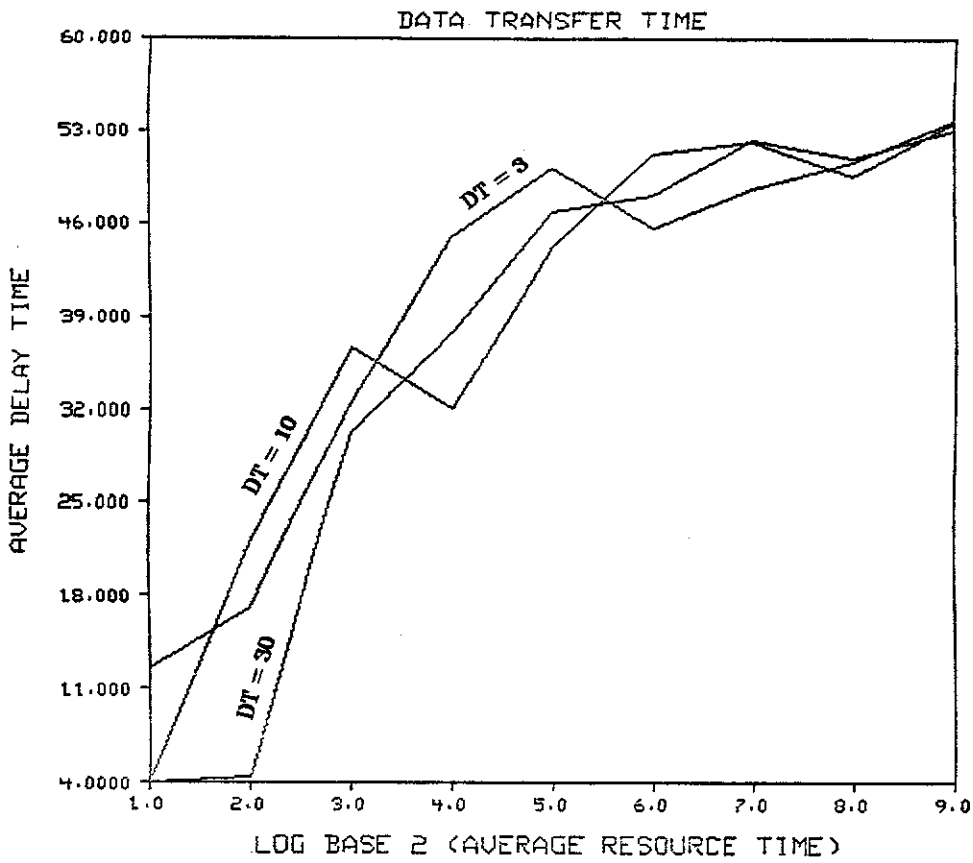


Figure 4.20 - Average Request Delay Time of the dynamic case of the distributed algorithm from the effects of Resource Time and Data Transfer Time

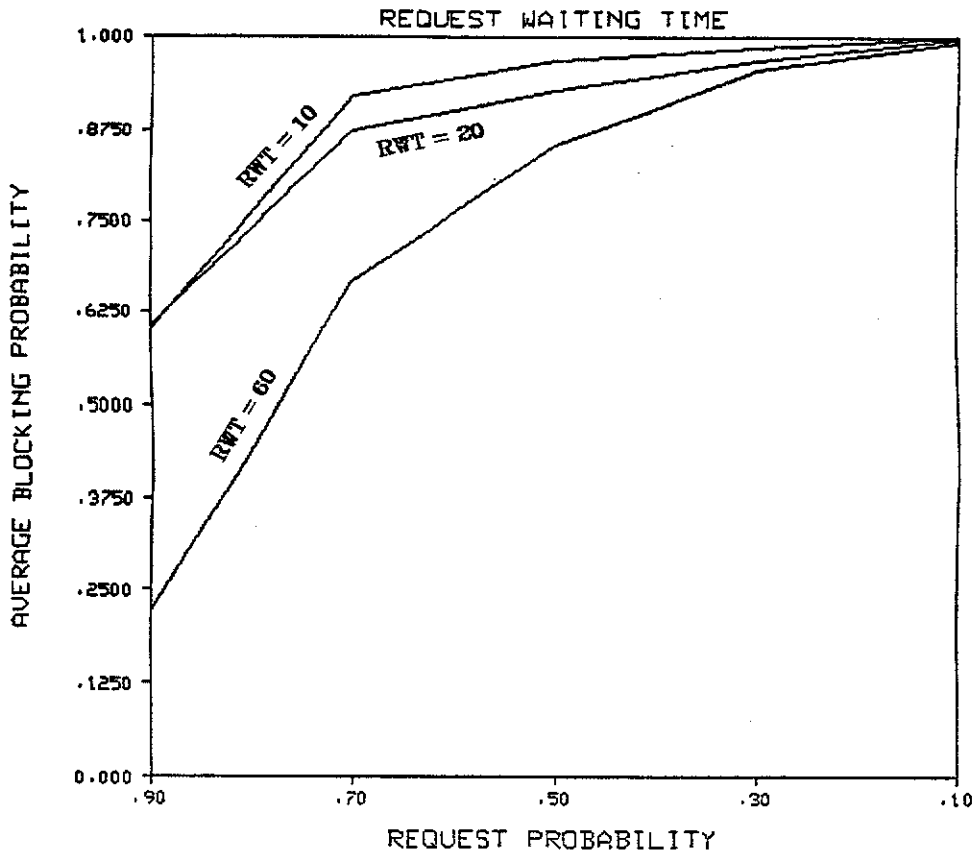


Figure 4.21 - Average Blocking Probability of the dynamic case of the distributed algorithm from the effects of Request Probability and Request Wait Time

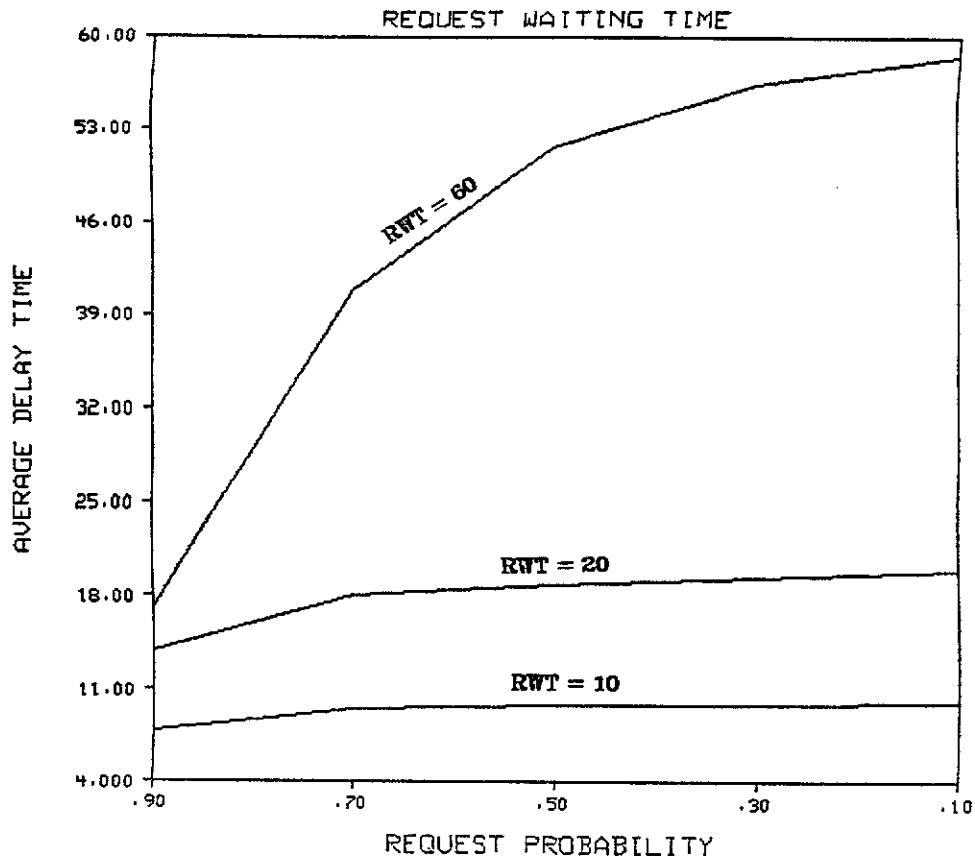


Figure 4.22 - Average Request Delay Time of the dynamic case of the distributed algorithm from the effects of Request Probability and Request Wait Time

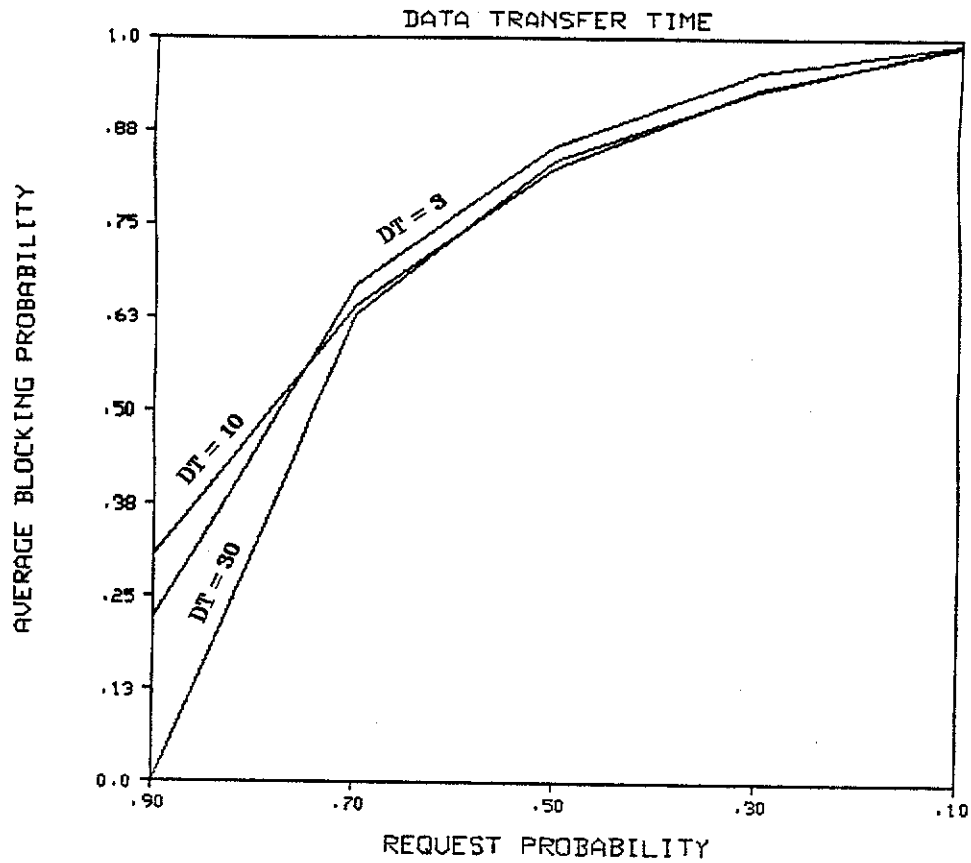


Figure 4.23 - Average Blocking Probability of the dynamic case of the distributed algorithm from the effects of Request Probability and Data Transfer Time

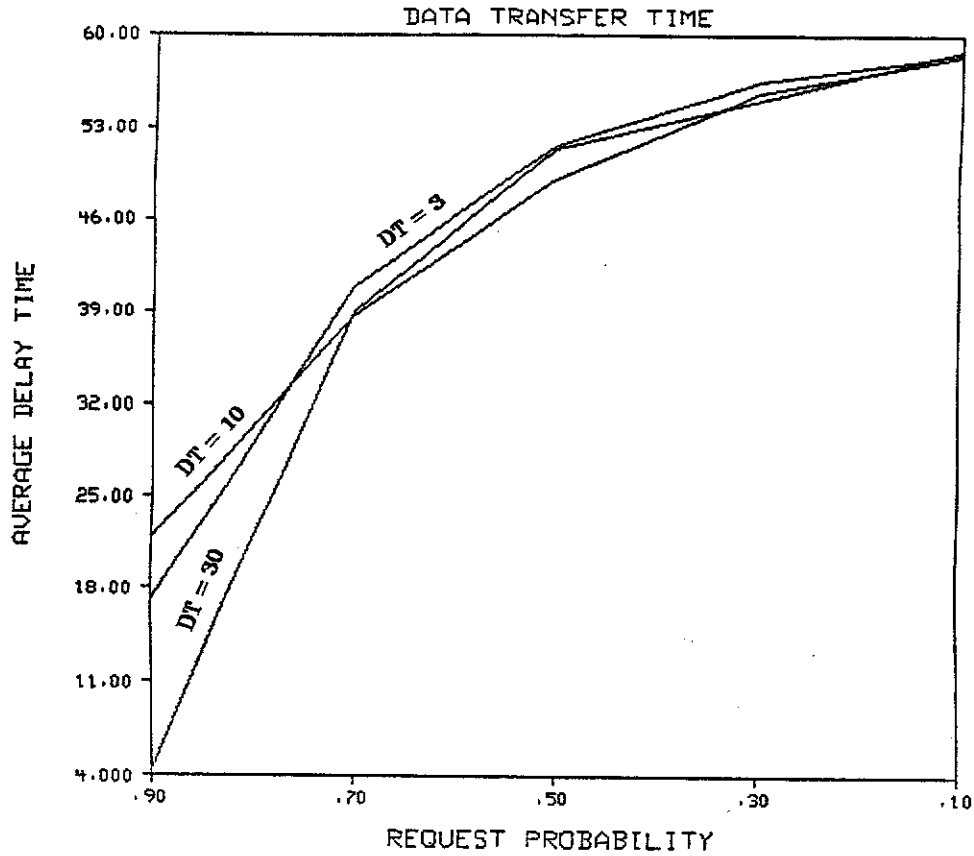


Figure 4.24 - Average Request Delay Time of the dynamic case of the distributed algorithm from the effects of Request Probability and Data Transfer Time

CHAPTER V - CONCLUSION AND EXTENSIONS.

In this thesis, a study of resources scheduling on multiple resources using interconnection networks was presented. The resource scheduling problem is different from the conventional address mapping problem on interconnection networks because a request is not directed to a particular destination address but to any one of a pool of free resources. Centralized scheduling algorithms are very inefficient because all the requesting processors must be serviced sequentially, and the central controller is a source of bottleneck when the number of requesting processors is large. The distributed scheduling algorithm allows the scheduling of all the processors to be performed in time proportional to the delay of the network.

An interconnection network for resource sharing may operate in two ways. First, the network is "circuit switched" and the processor and resource are continuously connected for the duration of use. In this case, the network may be partially busy when a new request is initiated. To avoid excessive blocking, the network should provide conflict-free access even when other connections are present. A distributed scheduling algorithm was designed for the cross-bar switch and the implementation of a cross-bar switch cell was presented. Each cell can be implemented with 12 gates and a flip-flop when the data path is one bit wide. The cell is designed with the minimum number of signal lines. As a result, the scheduling algorithm is divided into two phases and the algorithm is asymmetric, that is, priority is induced into the scheduling of multiple requests.

In order to allow the algorithm to operate in one phase, more signal lines are needed between adjacent cells as shown in the distributed Omega and Binary n-Cube networks. To remove the asymmetry in the scheduling, each resource can send a short pulse along the resource availability line. This pulse acts like a token and one of the requesting processors receiving this pulse will be allocated. When different resources issue tokens randomly, the algorithm is symmetric.

Second, the network is "packet switched" or operates in block-transfer mode. In this case, the resources are connected to the processors for a short duration of time and can relinquish the network after tasks are assigned. When a new set of requests are initiated, the network is almost or completely free. Networks with logarithmic delays are suitable for this application. An optimal centralized scheduling algorithm has been studied for the Omega and binary n-cube networks. It is shown that there is an average blocking probability of 1%. This means that these networks have behavior close to the cross-bar switch for resource sharing in a block transfer mode.

The centralized optimal algorithm has exponential time complexity. We studied, respectively, two centralized heuristics (with time complexities $0(N^2\log_2 N)$ and $0(N\log_2 N)$ and a distributed algorithm (with time complexity $0(\log_2 N)$). In the distributed algorithm for the static case, each exchange box in the network operates asynchronously and is responsible for resolving multiple requests directed to it. Resource availability information is also passed along the network to the processors. The control of the network can be hardwired or microprogrammed. The blocking probability increases as the time complexity decreases. In the worst case (distributed algorithm), the blocking probability is around 19%. In the distributed algorithm for the dynamic case,

multiple resources are used with varying processing time. The number of resources which is requested by a processor is a variable of geometric distribution. Other variables involved are the rate of job interarrival time, length of data transfer time once resources are allocated, and an allowable period of time during which a request may search for free resources. In addition, broadcasting capability was incorporated into each exchange box which allows multiple resources to be requested.

Several extensions can be included in the design. We discuss them briefly here.

(1) The resources connected on the network do not have to be identical. In a general system, there are multiple types of resources, each type is made up of a set of identical resources. The algorithms discussed have to be modified by identifying the type of resource requested by a processor and the type of resource available on a resource availability line. This can be done by sending a binary request code (instead of 1 bit) in the distributed algorithms. In the distributed Omega and binary n-cube networks, multiple resource availability registers have to be used in each exchange box.

(2) There is a tradeoff between the time complexity of the algorithm and the number of signal lines between two adjacent cells in the distributed RSIN's. A one-bit data path is assumed in the distributed cross-bar switch. In the distributed Omega and binary n-cube networks, parallel data path are assumed. This can be reduced by appropriate multiplexing at the external chip interface level.

The distributed algorithm implemented in each exchange box does not preclude operation under the address mapping mode. Further, the theory underlying the design of the distributed Omega and binary n-cube networks can be applied to other interconnection networks such as the Banyan [GOK73], and Delta [PAT81]. In these networks, the number of input and output ports can be different. The performance will be evaluated in the future. Future studies also include the performance evaluation of the dynamic case of the algorithm using other probability distributions.

REFERENCES

REFERENCES

[BAR81] G. H. Barnes and S. F. Lundstrom, "Design and Validation of a Connection Network for Many-Processor Multiprocessor Systems," *IEEE Computer,* Vol. 14, No. 12, pp. 31-41, Dec. 81.

[BAT74] K. E. Batcher, "STARAN Parallel Processing System Hardware," *Proc. of AFIPS 1974 National Computer Conf.,* Vol. 43, pp. 405-410, May 1974.

[BAT76] K. E. Batcher, "The Flip Network in STARAN," *Proc. of 1976 Int'l Conf. on Parallel Processing,* Michigan, pp. 65-71, 1976.

[BER82] R. Bernhard, "Computing at the speed limit", *IEEE Spectrum,* pp. 26-31, July 82.

[BUR79] Burroughs Corp., *Final Report, Numerical Aerodynamic Simulation Facility Feasibility Study,* NASA Contractor Reports CR152284 and CR152285, Burroughs Corp., Paoli, PA, March 1979.

[FEN74] T. Feng, "Data Manipulating Functions in Parallel Processors and Their Implications," *IEEE Trans. Computers,* Vol. C-23, No. 3, pp. 309-318, Mar. 1974.

[FRA80] M. A. Franklin, "VLSI Performance Comparison of Banyan and Cross-bar Communication Networks," *Proc. of Workshop on Interconnection Networks,* pp. 20-28, Apr. 1980.

[GOK73] L. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multi-processor Systems," *Proc. 1st Annual Comp. Architecture Conf.,* pp. 21-28, Dec. 1973.

[GOK76] L. R. Goke, *Banyan Networks for Partitioning Multiprocessor Systems* Ph.D. Thesis, Univ. of Florida, 1976.

[J&K80] B. Jayaraman and R. M. Keller, "Resource Control In A Demand-Driven Data Flow Model," *IEEE Computer Magazine,* 1980.

[JEN81] R. Jenevein, D. Degroot and G. J. Lipovski, "A Hardware Support Mechanism for Scheduling Resources in a Parallel Machine Environment," *Proc. of 8'th Annual Symposium on Computer Architecture,* pp. 57-66, May 1981.

[KIM75] S. R. Kimbleton, "Computer Communication Networks: Approaches, Objectives, and Performance Considerations," *Computing Surveys,* Vol. 7, No. 3, pp. 129-173, Sep. 75.

[KUC68] D. J. Kuck, "ILLIAC IV Software and Application Programming," *IEEE Trans. on Comp.,* Vol. C-17, pp. 746-757, Aug. 1968.

[LAM78] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM,* Vol. 21, No. 7, July 1978.

[LAW75] D. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers,* Vol. C-24, No. 12, pp. 215-255, Dec. 1975.

[M&R76] S. Mahmoud and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Networks," *Trans. on Data Base Sys.*, Vol. 1, No. 1, Mar. 76.

[McD70] M. H. McDougall, "Computer System Simulation: An Introduction," *Computing Surveys*, Vol. 2, No. 3, pp. 191-209, Sep. 79.

[MCD78] W. C. McDonald and J. M. Williams, "The Advanced Data Processing Test Bed," *Proc. of COMPSAC 78*, pp. 346-351, March 1978.

[ORN75] S. M. Ornstein et al., "Pluribus-A Reliable Multiprocessor," *Proc. AFIPS 1975 National Computer Conference*, AFIPS Press, Montvale, N.J., pp. 551-559, 1975.

[PAT81] J. H. Patel, "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Trans. on Computers*, Vol. C-20, No. 10, pp. 771-780, Oct. 1981.

[PEA77] M. C. Pease, "The Indirect Binary n-binary n-cube Microprocessor Array," *IEEE Trans. on Computers*, Vol. C-26, No. 5, pp. 458-473, May 1977.

[RAT80] B. D. Rathi, A. R. Tripathi and G. J. Lipovski, "Hardwired Resource Allocators for Reconfigurable Architectures," *Proc. of 1980 International Conference on Parallel Processing*, pp. 109-117, Aug. 1980.

[RUM77] J. Rumbaugh, "A Data Flow Multiprocessor," *IEEE Trans. on Computers*, Vol. C-26, No. 2, pp. 138-146, Feb. 77.

[SEJ80] M. C. Sejnowski et al., "Overview of the Texas Reconfigurable Array Computer," *AFIPS Conference Proceedings*, Vol. 49, pp. 631-642, 1980.

[SIE79] H. J. Siegel, "Interconnection Networks for SIMD Machines," *IEEE Computer Magazine*, Vol. 12, No. 6, pp. 57-60, June 1979.

[SIE81a] H. J. Siegel and R. J. McMillen, "The Cube Network as a Distributed Processing Test Bed Switch," *2nd Int'l. Conf. on Dist. Comp. Sys.*, pp. 377-387, April 1981.

[SIE81b] H. J. Siegel and R. J. McMillen, "Using the Augmented Data Manipulator Network in PASM," *IEEE Computer*, Vol. 14, No. 2, pp. 25-33, Feb. 1981.

[SIE81c] H. J. Siegel and R. J. McMillen, "The Multistage Cube: A Versatile Interconnection Network", ,ul IEEE Computer, Vol. 14, No. 12, pp. 65-76, Dec. 1981.

[STO71] H. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers*, Vol. C-20, No. 2, pp. 153-161, Feb. 1971.

[WAH82] B. W. Wah and A. H. Hicks, "Distributed Scheduling of resources on Interconnection Networks", *National Computer Conference*, Vol. 51, pp. 697-709, 1982.

[W&G82] I. Watson and J. Gord, "A Practical Data Flow Computer," *IEEE Computer Magazine*, pp. 51-57, Feb. 82.

[WU80] C. Wu and T. Y. Feng, "On a Class of Multistage Interconnection Networks," *IEEE Trans. on Computers*, Vol. C-29, No. 8, pp. 694-702, Aug. 1980.

[WUL72] W. A. Wulf and C. G. Bell, "C.mmp--A Multi-mini Processor," *Proc. AFIPS 1972 Fall Joint Comp. Conf.*, Vol. 41, AFIPS Press, Montvale, NJ, pp. 765-777, 1972.

APPENDIX

APPENDIX

This section includes all source code written for simulations of Scheduling Algorithms, Command file, Input file names, and Output results.

```
c*************************************************************************
c
c        *****  Unique Paths Table Generation  *****
c
c  Program name   :  Net4096.f
c  Language       :  Fortran 77
c  Programmer     :  Anthony Hicks
c  Facilities     :  UNIX Dual-Vax 780 on ECN
c                    Electrical Engineering Dept.
c                    Purdue University
c  Date written   :  4/15/1981
c  Date updated   :  2/20/1982
c
c  Description    :
c     Program to compute all 4096 possible connections of indirect
c      binary cube & omega network of 8 processors to 8 resources.
c     Every nested 'Do' in this program represents operations of
c       1 box out of 12 boxes in the network. They are numbered
c       starting with 1 and in column major fashion.
c
c  External Files -  For Cube network     Input  : In4096.cube
c                                         Output : Arrange.cub
c                    For Omega network    Input  : In4096.ohm
c                                         Output : Arrange.ohm
c
c*************************************************************************
c
c
      integer data(50),fstage(8),top(50),bottom(50),link(50)
      character*5 net
      data fstage/1,3,5,7,2,4,6,8/
      read(5,11) net
   11 format(a5)
      do 5 i = 1,32
    5 read(5,*) data(i), top(i), bottom(i)
      do 10 mode1 = 1,2
         mod1 = mode1 - 1
         call assign(mode1-1,top,bottom,link,1)
       do 10 mode2 = 1,2
          mod2 = mode2 -1
          call assign(mode2-1,top,bottom,link,3)
        do 10 mode3 = 1,2
           mod3 = mode3 - 1
           call assign(mode3-1,top,bottom,link,5)
         do 10 mode4 = 1,2
            mod4 = mode4 - 1
            call assign(mode4-1,top,bottom,link,7)
          do 10 mode5 = 1,2
             mod5 = mode5 -1
             call assign(mode5-1,top,bottom,link,9)
           do 10 mode6 = 1,2
              mod6 = mode6 - 1
              call assign(mode6-1,top,bottom,link,11)
            do 10 mode7 = 1,2
               mod7 = mode7 - 1
               call assign(mode7-1,top,bottom,link,13)
             do 10 mode8 = 1,2
                mod8 = mode8 - 1
                call assign(mode8-1,top,bottom,link,15)
              do 10 mode9 = 1,2
```

```
                  mod9 = mode9 -1
                  call assign(mode9-1, top, bottom, link, 17)
                do 10 mode10 = 1,2
                   mod10 = mode10 - 1
                   call assign(mode10-1, top, bottom, link, 19)
                 do 10 mode11 = 1,2
                    mod11 = mode11  -1
                    call assign(mode11-1, top, bottom, link, 21)
                  do 10 mode12 = 1,2
                     mod12 = mode12 - 1
                     call assign(mode12-1, top, bottom, link, 23)
                     if(net .eq. 'omega') then
                   write(6,20) (data(link(link(link(fstage(i))))), i=1,8)
c      *mod1, mod2, mod3, mod4, mod5, mod6, mod7, mod8, mod9, mod10, mod11, mod12
                     else
                      write(6,20) (data(link(link(link(i)))), i=1,8)
                     endif
   10 continue
   20 format(1x,8(1x,i1))
      stop
      end
c
c
c*****************************************
c
c   Routine to set boxes for connections
c
c    Convention:   '0' for Straight.
c                  '1' for Diagonal.
c
c*****************************************
c
c
      subroutine assign(mode, top, bottom, link, index)
      integer top(50), bottom(50), link(50)
      if(mode .eq. 0) go to 10
      link(index) = bottom(index)
      link(index+1) = top(index+1)
      go to 20
   10 link(index) = top(index)
      link(index+1) = bottom(index+1)
   20 return
      end
```

```
program tree(input,output,arrange);
(*************************************************************)
(*                                                           *)
(*          Tilted Binary Tree Creation Program              *)
(*                                                           *)
(*    Program name:    Tree.p                                *)
(*                                                           *)
(*    Language     :   Pascal                                *)
(*                                                           *)
(*    Description :    To create a tree structure for        *)
(*                        a cube interconnection network     *)
(*                        to be search as a centralized      *)
(*                        algorithm                          *)
(*                                                           *)
(*    Programmer   :   Anthony Hicks                         *)
(*                                                           *)
(*    Files        :   External Input  :  Arrange            *)
(*                             Output :  ########            *)
(*                                                           *)
(*    System       :   Unix v7.0 on Vax 780                  *)
(*                     EE Dept. Purdue University            *)
(*                                                           *)
(*    Date written:    9/11/1981       Updated : 9/30/1981  *)
(*                                                           *)
(*************************************************************)

const       numofnod = 35000;
            limit    =    8;

type        trees    = record
                         resourc : integer;
                         side    : integer;
                         down    : integer
                       end;

var          addloca  : integer;    (* location a node to be added *)
             arrange  : text;       (* 4096 connections            *)
             avail    : integer;    (* next avail node             *)
             endofdat : boolean;    (* end of input data flag      *)
             i,j,k    : integer;    (* general counter variables   *)
             lastloca : integer;    (* last location o a root      *)
             level    : integer;    (* level counter of nodes      *)
             locapntr : integer;    (* present location of pointer *)
             newnode  : integer;    (* new input node              *)
             node     : array[1..numofnod] of trees;
             processo : array[1..limit] of integer;
             resou    : array[1..limit] of integer;

procedure addanode; forward;
procedure buildit;  forward;
procedure builtree; forward;
procedure fileit;   forward;
procedure finsert;  forward;
procedure getinput; forward;
procedure initiali; forward;
procedure insert;   forward;
```

```
(*****************************************************************)
(*                                                               *)
(*    Addanode - Routine to add a node to the tree               *)
(*                                                               *)
(*****************************************************************)
procedure addanode;
     begin
          case addloca of
               1: node[locapntr].side := avail;
               2: node[locapntr].down := avail;
          end;     (* case of *)
          lastloca := locapntr;
          locapntr := avail;
          avail    := node[avail].side;
          node[locapntr].side := -1;
          node[locapntr].resourc := newnode;
     end;

(*****************************************************************)
(*                                                               *)
(*    Buildit - Routine to build the whole tree                  *)
(*                                                               *)
(*****************************************************************)
procedure buildit;
     begin
          while not endofdat do
               begin
                    locapntr := 1;
                    level := 0;
                    for i:= 1 to limit do
                         begin
                              level := level + 1;
                              newnode := resou[processo[level]];
                              builtree;
                         end;
                    for j:= 1 to limit do
                         read(arrange,resou[j]);
                    readln(arrange);
                    if eof(arrange) then endofdat := true;
               end;
          node[1].resourc := avail;
     end;


(*****************************************************************)
(*                                                               *)
(*    Builtree - Routine to add or insert a node to the tree     *)
(*                                                               *)
(*****************************************************************)
procedure builtree;
     begin
          if node[locapntr].down = -1 then
               begin
                    addloca := 2;
                    addanode;
               end
          else
               begin
                    lastloca := locapntr;
                    locapntr := node[locapntr].down;
```

```
                if newnode < node[locapntr].resourc then
                    finsert;
                if newnode > node[locapntr].resourc then
                    begin
                        if node[locapntr].side = -1 then
                            begin
                                addloca := 1;
                                addanode;
                            end
                        else
                            begin
                                repeat
                                    lastloca := locapntr;
                                    locapntr := node[locapntr].side;
                                until (newnode <= node[locapntr].resourc)
                                    or (node[locapntr].side = -1);
                                if newnode < node[locapntr].resourc then
                                    insert;
                                if node[locapntr].side = -1 then
                                    begin
                                        addloca := 1;
                                        addanode;
                                    end;
                            end;
                    end;
            end;
    end;

(*********************************************************************)
(*                                                                   *)
(*   Fileit - Routine to print out trees                             *)
(*                                                                   *)
(*********************************************************************)
procedure fileit;
    begin
        for k:= 1 to avail do
            begin
            write(k:5,' ',node[k].resourc:1,' ',node[k].side:5,
                        ' ',node[k].down:5);
            writeln;
            end;
    end;

(*********************************************************************)
(*                                                                   *)
(*   Finsert - Routine to insert a node a a head level node          *)
(*                                                                   *)
(*********************************************************************)
procedure finsert;
    var save : integer;
    begin
        save := node[avail].side;
        node[avail].side := node[lastloca].down;
        node[lastloca].down := avail;
        locapntr := avail;
        avail := save;
        node[locapntr].resourc := newnode;
    end;
```

```
(*****************************************************************)
(*                                                               *)
(*    Getinput - Routine to read in input data                   *)
(*                                                               *)
(*****************************************************************)
procedure getinput;
     var m : integer;
     begin
          for m := 1 to limit do
             begin
              read(processo[m]);
              write(processo[m]:2);
             end;
          writeln;
          readln;
          reset (arrange);
          for m:= 1 to limit do
              read(arrange,resou[m]);
          readln(arrange);
     end;


(*****************************************************************)
(*                                                               *)
(*    Initiali - Routine to initialize the tree array            *)
(*                                                               *)
(*****************************************************************)
procedure initiali;
     var l : integer;
     begin
          for l := 1 to numofnod - 1 do
              begin
                  node[l].side := l+1;
                  node[l].down := -1;
              end;
          node[numofnod].side := -1;
          node[numofnod].down := -1;
          avail := 2;
          node[1].side := -1;
          endofdat := false;
     end;


(*****************************************************************)
(*                                                               *)
(*    Insert - Routine to insert a node on the same level        *)
(*                                                               *)
(*****************************************************************)
procedure insert;
     var save : integer;
     begin
          save := node[avail].side;
          node[avail].side := node[lastloca].side;
          node[lastloca].side := avail;
          locapntr := avail;
          avail := save;
          node[locapntr].resourc := newnode;
     end;
```

```
(*****************************************************************)
(*                                                               *)
(*    Driver - Main program                                      *)
(*                                                               *)
(*****************************************************************)
begin
     initiali;
     getinput;
     buildit;
     fileit;
end.
```

```
c*********************************************************************
c                                                                     *
c   B i n a r y   t r e e   S e a r c h   A l g o r i t h m           *
c                                                                     *
c  Program name   :  Search.f                                         *
c  Language       :  Fortran 77                                       *
c  Programmer     :  Anthony Hicks                                    *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                         *
c                    Electrical Engineering Dept.                     *
c                    Purdue University                                *
c  Date written   :  09/15/81                                         *
c  Date updated   :  02/20/82                                         *
c                                                                     *
c  Description    :                                                   *
c     This program search binary trees to compute the lower           *
c        bound blocking probabilities of the centralized              *
c        algorithm for the Cube & Omega network.                      *
c                                                                     *
c  External Files  -  Input   : Procreq   Requesting processors       *
c                               Resoava   Available resources         *
c                               Statable  Statistics table            *
c                               ########  Tree to be searched         *
c                     Output   : Statable  Statistics table           *
c                                                                     *
c*********************************************************************
c
c
      integer*2 size,data(30000),side(30000),down(30000),proc(8)
      read(5,*) (proc(i),i=1,8)
      read(5,*) a, size, side(1), down(1)
      call driver(size,proc,data,side,down)
      stop
      end
c
c          **********  D R I V E R  **********
c
      subroutine driver(size,proc,data,side,down)
      integer*2 size,data(size),side(size),down(size),
     *proces(8),resour(8),proc(8),sucpro,i,j,k,pnum,rnum
      integer statab(8,8,3)
      data(1) = size
      open(2,file='procreq',status='old')
      open(3,file='resoava',status='old')
      open(4,file='statable',status='old')
      rewind 2
      rewind 3
      rewind 4
    3 read(5,*,end=7) n, data(n), side(n), down(n)
      go to 3
    7 read(4,*) hello
      do 10 k=1,3
          do 10 i=1,8
   10     read(4,*) (statab(i,j,k),j=1,8)
      close(4,status='keep')
      open(4,file='statable',status='old')
      rewind 4
  100 read(2,*,end=900) pnum, (proces(i),i=1,pnum)
  200 read(3,*,end=800) rnum, (resour(i),i=1,rnum)
      call search(size,sucpro,data,side,down,proces,resour,proc,
     *pnum,rnum)
```

```
      statab(pnum,rnum,1) = statab(pnum,rnum,1)+sucpro
      statab(pnum,rnum,2) = statab(pnum,rnum,2)+1
      statab(pnum,rnum,3) = statab(pnum,rnum,3)+sucpro**2
      go to 200
  800 close(3,status='keep')
      open(3,file='resoava',status='old')
      rewind 3
      go to 100
  900 write(4,17) (proc(k),k=1,8)
   17 format(2x,8i1)
      do 901 k=1,3
          do 901 i=1,8
  901     write(4,910) (statab(i,j,k),j=1,8)
  910 format(3x,8(2x,i8))
      endfile 4
      close(2,status='keep')
      close(3,status='keep')
      close(4,status='keep')
      return
      end
c
c      ********** S E A R C H **********
c
      subroutine search(size,sucpro,data,side,down,proces,resour,
     *proc,pnum,rnum)
      integer*2 size,data(size),side(size),down(size),pnum,
     *rnum,proces(8),resour(8),sucpro,stklev(50),stack1(50),
     *stkpno(50),proc(8),stkptr(50),ptr1,levcnt,laspno
      integer treptr, ptr2
      levcnt = 0
      laspno = 0
      sucpro = 0
      treptr = 1
      ptr1 = 0
      ptr2 = 0
  100 treptr = down(treptr)
      levcnt=levcnt+1
  200 if(side(treptr) .ne. -1) then
          ptr2 = ptr2 + 1
          stack1(ptr2) = side(treptr)
          stklev(ptr2) = levcnt
          stkptr(ptr2) = ptr1
          stkpno(ptr2) = sucpro
      endif
      if(proc(levcnt) .eq. proces(ptr1+1)) then
          ptr1 = ptr1 + 1
          do 5 io=1,rnum
              if(data(treptr) .eq. resour(io)) then
                  sucpro = sucpro + 1
                  go to 300
              endif
    5     continue
      endif
  300 if(sucpro .eq. pnum) go to 500
      if(down(treptr) .eq. -1 .or. ptr1 .eq. pnum) then
          if(ptr2 .eq. 0) go to 400
          treptr = stack1(ptr2)
          levcnt = stklev(ptr2)
          ptr1 = stkptr(ptr2)
          if(sucpro .gt. laspno) laspno = sucpro
```

```
            sucpro = stkpno(ptr2)
            ptr2 = ptr2 - 1
            go to 200
        else
            go to 100
        endif
400     if(sucpro .lt. laspno) sucpro = laspno
500     return
        end
```

```
c*********************************************************************
c                                                                     *
c        Centralized control Heuristic without retry                  *
c                                                                     *
c  Program name   :  Algo1.f                                          *
c  Language       :  Fortran 77                                       *
c  Programmer     :  Anthony Hicks                                    *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                         *
c                    Electrical Engineering Dept.                     *
c                    Purdue University                                *
c  Date written   :  10/24/81                                         *
c  Date updated   :  02/06/82                                         *
c                                                                     *
c  Description    :                                                   *
c      An allocation heuristic using direct matching between        *
c          processors & resources " without retry ".                  *
c                                                                     *
c  External Files  -  Input   : Procreq   Requesting processors       *
c                               Resoava   Available resources         *
c                               Indirect.cub  Connection table        *
c                          or   Indirect.ohm      "        "         *
c                      Output  : Statable  Statistics table           *
c                                                                     *
c*********************************************************************
c
      character*5 net
      integer data(50),top(50),bottom(50),link(50),
     *proces(8),resour(8),sucpro,pnum,rnum,shfnum
      integer statab(8,8,3),fstage(8),addres(8,3)
      common addres,data,top,bottom,link,proces,resour,
     *pnum,rnum
      data fstage/1,3,5,7,2,4,6,8/
      open(2,file='procreq',status='old')
      open(3,file='resoava',status='old')
      open(4,file='statable',status='old')
      rewind 2
      rewind 3
      rewind 4
      read(5,1) net
    1 format(a5)
      read(5,*) shift,shfnum
      do 6 i=1,8
    6 read(5,*) (addres(i,j),j=1,3)
      do 7 i=1,32
    7 read(5,*) data(i),top(i),bottom(i)
  100 read(2,*,end=900) pnum, (proces(i),i=1,pnum)
          write(6,*) 'request: ',(proces(kk),kk=1,pnum)
      if(net.eq.'omega') then
          do 11 i=1,pnum
   11     proces(i) = fstage(proces(i))
          endif
  200 read(3,*,end=800) rnum, (resour(i),i=1,rnum)
          write(6,*) 'resoava: ',(resour(i),i=1,rnum)
          write(6,*) 'mapped to: ',(proces(kk),kk=1,pnum)
      call algone(sucpro,shift,shfnum)
      statab(pnum,rnum,1) = statab(pnum,rnum,1)+sucpro
      statab(pnum,rnum,2) = statab(pnum,rnum,2)+1
      statab(pnum,rnum,3) = statab(pnum,rnum,3)+sucpro**2
      go to 200
  800 close(3,status='keep')
```

```
      open(3,file='resoava',status='old')
      rewind 3
      go to 100
  900 do 901 k=1,3
          do 901 i=1,8
  901     write(4,910) (statab(i,j,k),j=1,8)
  910 format(3x,8(2x,i8))
      endfile 4
      close(2,status='keep')
      close(3,status='keep')
      close(4,status='keep')
      stop
      end
c
c**********************************
c
c  Direct matching algorithm
c
c**********************************
c
      subroutine algone(sucpro,shift,shfnum)
      integer flag(12,2),addres(8,3),sucpro,pnum,rnum,
     *data(50),top(50),bottom(50),link(50),proces(8),
     *resour(8),shift,tempro,box,save,shfnum,bokeep(3)
      common addres,data,top,bottom,link,proces,resour,
     *pnum,rnum
      do 2 i=1,12
         do 2 j=1,2
    2 flag(i,j) = 0
      sucpro = 0
      if(shift .eq. 0) go to 20
      do 3 m=1,shfnum
      save = resour(1)
      do 10 i=1,rnum-1
   10 resour(i) = resour(i+1)
    3 resour(rnum) = save
   20 npnum = pnum
      if(pnum .gt. rnum) npnum = rnum
      do 30 i=1,npnum
      tempro = proces(i)
         do 35 k=1,3
         call conver(tempro,addres(resour(i),k),iset)
         box = (tempro+1)/2
         now = box*2-1
         if(flag(box,1) .eq. 0)  then
             call assign(iset,top,bottom,link,now)
             bokeep(k) = box
             flag(box,2) = iset
         endif
          write(6,*) 'box: ',box,flag(box,1),flag(box,2)
      if(flag(box,1) .eq. 1) then
         if(flag(box,2) .ne. iset)go to 30
         endif
   35 tempro = link(tempro)
      if(data(link(link(link(proces(i))))) .eq. resour(i)) then
         sucpro = sucpro + 1
         do 113 k1=1,3
  113    flag(bokeep(k1),1) = 1
       write(6,*) 'succeeded: ',proces(i),' to ',resour(i)
         endif
```

```
   30 continue
      return
      end
c
c*********************************
c
c  Box function set routine
c
c*********************************
c
      subroutine assign(mode,top,bottom,link,index)
      integer top(50), bottom(50), link(50)
      if(mode .eq. 0) go to 10
      link(index) = bottom(index)
      link(index+1) = top(index+1)
      go to 20
   10 link(index) = top(index)
      link(index+1) = bottom(index+1)
   20 return
      end
c
c***********************************
c
c  Box input terminal  conversions
c
c***********************************
c
      subroutine conver(inport,jset,newset)
      integer inport,jset,newset
      if(inport/2*2 .eq. inport) then
         if(jset .eq. 0) newset = 1
         if(jset .eq. 1) newset = 0
         else
            newset = jset
         endif
      return
      end
```

```
c*********************************************************************
c                                                                     *
c          Centralized control Heuristic with RETRY                   *
c                                                                     *
c  Program name   :  Algo2.f                                          *
c  Language       :  Fortran 77                                       *
c  Programmer     :  Anthony Hicks                                    *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                         *
c                    Electrical Engineering Dept.                     *
c                    Purdue University                                *
c  Date written   :  10/24/81                                         *
c  Date updated   :  02/04/82                                         *
c                                                                     *
c  Description    :                                                   *
c     An allocation heuristic by direct matching between             *
c          processors & resources then RETRY all other               *
c          available resources if it failed to be connected          *
c          the 1st time until there is a connection or all           *
c          of resources have been tried.                             *
c                                                                     *
c  External Files  -  Input   : Procreq   Requesting processors       *
c                               Resoava   Available resources         *
c                               Indirect.cub  Connection table        *
c                         or    Indirect.ohm      "       "           *
c                       Output  : Statable  Statistics table          *
c                                                                     *
c*********************************************************************
c
c
      character*5 net
      integer data(50),top(50),bottom(50),link(50),
     *proces(8),resour(8),sucpro,pnum,rnum,shfnum
      integer statab(8,8,3),fstage(8),addres(8,3)
      common addres,data,top,bottom,link,proces,resour,
     *pnum,rnum
      data fstage/1,3,5,7,2,4,6,8/
      open(2,file='procreq',status='old')
      open(3,file='resoava',status='old')
      open(4,file='statable',status='old')
      rewind 2
      rewind 3
      rewind 4
      read(5,1) net
    1 format(a5)
      read(5,*) shift,shfnum,limit
      do 6 i=1,8
    6 read(5,*) (addres(i,j),j=1,3)
      do 7 i=1,32
    7 read(5,*) data(i),top(i),bottom(i)
  100 read(2,*,end=900) pnum, (proces(i),i=1,pnum)
      if(net.eq.'omega') then
         do 11 i=1,pnum
   11    proces(i) = fstage(proces(i))
         endif
  200 read(3,*,end=800) rnum, (resour(i),i=1,rnum)
      call algtwo(sucpro,shift,shfnum,limit)
      statab(pnum,rnum,1) = statab(pnum,rnum,1)+sucpro
      statab(pnum,rnum,2) = statab(pnum,rnum,2)+1
      statab(pnum,rnum,3) = statab(pnum,rnum,3)+sucpro**2
      go to 200
```

```
  800 close(3,status='keep')
      open(3,file='resoava',status='old')
      rewind 3
      go to 100
  900 do 901 k=1,3
          do 901 i=1,8
  901     write(4,910) (statab(i,j,k),j=1,8)
  910 format(3x,8(2x,i8))
      endfile 4
      close(2,status='keep')
      close(3,status='keep')
      close(4,status='keep')
      stop
      end
c
c*****************************************
c
c  Direct matching algorithm with retry
c
c*****************************************
c
      subroutine algtwo(sucpro,shift,shfnum,limit)
      integer flag(12,2),addres(8,3),sucpro,pnum,rnum,
     *data(50),top(50),bottom(50),link(50),proces(8),
     *resour(8),shift,tempro,box,save,shfnum,rstatu(8)
      common addres,data,top,bottom,link,proces,resour,
     *pnum,rnum
      integer pointr,bokeep(3)
      do 1 i=1,rnum
    1 rstatu(i) = 0
      do 2 i=1,12
         do 2 j=1,2
    2 flag(i,j) = 0
      sucpro = 0
      if(shift .eq. 0) go to 20
      do 3 m=1,shfnum
      save = resour(1)
      do 10 i=1,rnum-1
   10 resour(i) = resour(i+1)
    3 resour(rnum) = save
   20 npnum = pnum
      if(pnum .gt. rnum .and. limit .le.1) npnum = rnum
      pointr = 1
      do 30 i=1,npnum
         if(pointr.gt.rnum) go to 30
         do 34 l=1,rnum
         ko = pointr+l-1
         if(ko.gt.rnum) ko= ko-rnum
         if(rstatu(ko) .eq.1) then
              if(ko.eq.pointr)pointr = pointr+1
              go to 34
              endif
         tempro = proces(i)
         do 35 k=1,3
         call conver(tempro,addres(resour(ko),k),iset)
         box = (tempro+1)/2
         now = box*2-1
         if(flag(box,1) .eq. 0)  then
              call assign(iset,top,bottom,link,now)
              bokeep(k) = box
```

```
            flag(box,2) = iset
         endif
      if(flag(box,1) .eq. 1) then
         if(flag(box,2) .ne. iset)go to 34
         endif
   35 tempro = link(tempro)
      if(data(link(link(link(proces(i))))) .eq. resour(ko)) then
         sucpro = sucpro + 1
         rstatu(ko) = 1
         do 113 ll=1,3
  113    flag(bokeep(ll),1) = 1
         if(l.eq.1) pointr = pointr+1
         go to 30
         endif
   34 continue
   30 continue
      return
      end
c
c************************************
c
c  Box function set routine
c
c************************************
c
      subroutine assign(mode,top,bottom,link,index)
      integer top(50), bottom(50), link(50)
      if(mode .eq. 0) go to 10
      link(index) = bottom(index)
      link(index+1) = top(index+1)
      go to 20
   10 link(index) = top(index)
      link(index+1) = bottom(index+1)
   20 return
      end
c
c************************************
c
c  Box input terminal  conversions
c
c************************************
c
      subroutine conver(inport,jset,newset)
      integer inport,jset,newset
      if(inport/2*2 .eq. inport) then
         if(jset .eq. 0) newset = 1
         if(jset .eq. 1) newset = 0
         else
             newset = jset
         endif
      return
      end
```

```
c*****************************************************************
c                                                                 *
c        D I S T R I B U T E D    C O N T R O L                   *
c                    A L G O R I T H M                            *
c                                                                 *
c  Program name   :  Distritime.f                                 *
c  Language       :  Fortran 77                                   *
c  Programmer     :  Anthony Hicks                                *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                     *
c                    Electrical Engineering Dept.                 *
c                    Purdue University                            *
c  Date written   :  11/05/81                                     *
c  Date updated   :  01/26/82                                     *
c                                                                 *
c  Description    :                                              *
c      This program gives each of the control box intelligence    *
c          to resolve incoming requests and routes them to        *
c          next stage & also keep track of the average delay      *
c          for each request to either made it or be rejected.     *
c                                                                 *
c  External Files  -  Input    : Procreq   Requesting processors  *
c                                Resoava   Available resources    *
c                                Indist.cube  Connection table    *
c                         or     Indist.ohm        "       "      *
c                       Output  : Statable  Statistics table      *
c                                                                 *
c*****************************************************************
c
c
c       D e c l a r a t i o n s
c
      integer data(50),btop(50),bbot(50),flink(50),blink(50),
     *proces(8),resour(8),sucpro,pnum,rnum,fbot(50),ftop(50)
      integer statab(8,8,3),flag(12,2),box(12,5)
      integer count,pronum(260),pin(260,8),rin(260,8),
     *resnum(260)
      double precision time,stat(8,8,2)
      character*5 net
      common ftop,fbot,btop,bbot,flink,blink,proces,resour,
     *pnum,rnum,flag,box,iptra,iptrb,time
c
c       I n i t i a l i z a t i o n s
c
      data statab,stat/192*0,128*0./
      open(2,file='procreq',status='old')
      open(3,file='resoava',status='old')
      open(4,file='statable',status='old')
      rewind 2
      rewind 3
      rewind 4
      do 1 la=1,12
      box(la,1) = la*2+7
    1 box(la,2) = box(la,1)+1
c
c     I n p u t   n e t w o r k   c o n n e c t i o n s
c
      read(5,2) net
    2 format(a5)
      do 7 i=1,8
```

```
    7 read(5,*) data(i),ftop(i),fbot(i),btop(i),bbot(i)
      do 8 i=9,31,2
      read(5,*) data(i),ftop(i),fbot(i),btop(i),bbot(i)
      data(i+1) = data(i)
      ftop(i+1) = ftop(i)
      fbot(i+1) = fbot(i)
      btop(i+1) = btop(i)
      bbot(i+1) = bbot(i)
    8 continue
      do 9 i=33,40
    9 read(5,*) data(i),ftop(i),fbot(i),btop(i),bbot(i)
      count = 1
c
c   I n p u t    C o m b i n a t i o n s
c
  100 read(2,*,end=200) pronum(count),
     *(pin(count,k),k=1,pronum(count))
      count = count + 1
      go to 100
  200 kount = 1
  300 read(3,*,end=400) resnum(kount),
     *(rin(kount,k),k=1,resnum(kount))
      kount = kount + 1
      go to 300
c
c    G e n e r a t i n g    R e q u e s t s
c
  400 do 401 i=1,count-1
          pnum = pronum(i)
          do 11 io=1,pnum
   11     proces(io) = pin(i,io)
c
c    G e n e r a t i n g    A v a i l a b i l i t i e s
c
          do 401 j=1,kount-1
              rnum = resnum(j)
              do 12 jo=1,rnum
   12         resour(jo) = rin(j,jo)
c
c    B r o a d c a s t i n g
c
      call brocas(net,rnum,resour,box,btop,bbot,flag)
c
c    D i s t r i b u t e d    a l g o r i t h m
c
      call protoc(sucpro,net)
c
c    U p d a t i n g    s t a t i s t i c s
c
      statab(pnum,rnum,1) = statab(pnum,rnum,1)+sucpro
      statab(pnum,rnum,2) = statab(pnum,rnum,2)+1
      statab(pnum,rnum,3) = statab(pnum,rnum,3)+sucpro**2
      time = time/float(pnum)
      stat(pnum,rnum,1) = stat(pnum,rnum,1) + time
      stat(pnum,rnum,2) = stat(pnum,rnum,2) + time**2
  401 continue
c
c    O u t p u t    s t a t i s t i c s
c
      do 901 k=1,3
```

```
            do 901 i=1,8
  901       write(4,910) (statab(i,j,k),j=1,8)
        do 902 k=1,2
            do 902 i=1,8
  902       write(4,911) (stat(i,j,k),j=1,8)
  910 format(3x,8(2x,i8))
  911 format(2x,8(1x,f15.9))
c
c    C l o s i n g     f i l e s
c
        endfile 4
        close(2,status='keep')
        close(3,status='keep')
        close(4,status='keep')
        stop
        end
c
c
c*********************************
c  Box function set routine
c*********************************
c
c
        subroutine assign(mode,top,bottom,flink,blink,index)
        integer top(50), blink(50), bottom(50), flink(50)
        if(mode .eq. 0) go to 10
        flink(index) = bottom(index)
        flink(index+1) = top(index+1)
        blink(bottom(index)) = index
        blink(top(index+1)) = index + 1
        go to 20
   10   flink(index) = top(index)
        flink(index+1) = bottom(index+1)
        blink(top(index)) = index
        blink(bottom(index+1)) = index + 1
   20   return
        end
c
c
c***********************************************
c     Routine to Broadcast available resources
c***********************************************
c
c
        subroutine brocas(net,rnum,resour,box,btop,bbot,flag)
        integer rnum,box(12,5),avail,btop(50),bbot(50),resour(8),
       *flag(12,2)
        character*5 net
        do 2 i=1,12
           do 3 k=3,5
    3      box(i,k) = 0
           flag(i,1) = 2
    2      flag(i,2) = 2
          do 10 i=1,rnum
           avail = resour(i) + 32
           if(net .eq. 'cube ') then
              if(avail.ge.33 .and. avail.le.36) avail=btop(avail)
              if(avail.ge.37 .and. avail.le.40) avail=bbot(avail)
           endif
           if(net .eq. 'omega') then
```

```
            if(avail/2*2 .ne. avail)avail = btop(avail)
            if(avail/2*2 .eq. avail)avail = bbot(avail)
         endif
         ijk=(avail-7)/2
         if(avail.eq.box(ijk,1)) then
            box(ijk,3) = 1
            box(ijk,5) = box(ijk,5) + 1
         endif
         if(avail.eq.box(ijk,2)) then
            box(ijk,4) = 1
            box(ijk,5) = box(ijk,5) + 1
         endif
 10      continue
      do 19 j=1,2
         do 20 k=9,12
         jo = k-((j-1)*4)
         call broset(box,btop,bbot,box(jo,1),box(jo,5))
 20      continue
 19   continue
      return
      end
c
c
c*****************************************************
c  Routine to set boxes for Broadcasting uses
c*****************************************************
c
c
      subroutine broset(box,btop,bbot,avail,sum)
      integer sum,btop(50),bbot(50),link(2),box(12,5),
     *curbox(2),avail
      link(1) = btop(avail)
      link(2) = bbot(avail)
      curbox(1) = (link(1)-7)/2
      curbox(2) = (link(2)-7)/2
 56   format(2x,4(2x,i5))
      do 11 j=1,2
         if(link(j).eq.box(curbox(j),1)) then
             box(curbox(j),3) = sum
             box(curbox(j),5) = box(curbox(j),5) + sum
         endif
         if(link(j).eq.box(curbox(j),2)) then
             box(curbox(j),4) = sum
             box(curbox(j),5) = box(curbox(j),5) + sum
         endif
 11   continue
      return
      end
c
c
c*****************
c  Sort Routine
c*****************
c
c
      subroutine sort(x,n)
      integer x(n)
      do 10 m=2,n
      k = m
 5    if(x(k) .le. x(k/2))go to 10
```

```
      ky= x(k)
      x(k) = x(k/2)
      x(k/2) = ky
      k= k/2
      if(k.gt.1) go to 5
   10 continue
      do 30 nm=2,n
      m = n-nm + 2
      itop = x(1)
      j = 1
   25 if(m.le.j*2) go to 27
      if(x(2*j).ge.x(2*j+1))i=2*j
      if(x(2*j).lt.x(2*j+1))i=2*j+1
      if(x(i).le.x(m))go to 27
      x(j) = x(i)
      j=i
      go to 25
   27 x(j) = x(m)
   30 x(m) = itop
      return
      end
c
c
c**********************************
c  Routine to make connections
c**********************************
c
c
      subroutine connec(t,b,mode,termi,sucpro,curbox,stackb)
      integer t,b,mode,termi,sucpro,stackb(30),flag(12,2),curbox,
     *box(12,5),ftop(50),btop(50),fbot(50),bbot(50),flink(50),
     *blink(50),proces(8),resour(8)
      double precision time
      common ftop,fbot,btop,bbot,flink,blink,proces,resour,
     *pnum,rnum,flag,box,iptra,iptrb,time
      if(t.eq.1 .and. mode.eq.0)flag(curbox,1) = 0
      if(t.eq.1 .and. mode.eq.1)flag(curbox,1) = 1
      if(b.eq.1 .and. mode.eq.0)flag(curbox,2) = 0
      if(b.eq.1 .and. mode.eq.1)flag(curbox,2) = 1
      if(t.eq.1) then
           call update(termi,sucpro,iptrb,iptra,flink,stackb,time)
           if(b.eq.1)call update(termi+1,sucpro,iptrb,iptra,flink,
     *stackb,time)
           endif
      if(b.eq.1 .and. t.ne.1)call update(termi,sucpro,iptrb,iptra,
     *flink,stackb,time)
      return
      end
c
c
c*****************************************************
c  Routine to update succesful processors counter
c*****************************************************
c
c
      subroutine update(termi,sucpro,iptrb,iptra,flink,stackb,time)
      double precision time
      integer termi,sucpro,iptrb,iptra,flink(50),stackb(30)
      if(termi.ge.25 .and. termi.le.32) then
        if(flink(termi).ge.33 .and. flink(termi).le.40)sucpro=sucpro+1
```

```
        else
             iptrb = iptrb+1
             time = time + 1.
             stackb(iptrb) = flink(termi)
        endif
        iptra =  iptra - 1
        return
        end
c
c
c*************************************************
c  Routine to backtrack whenever required
c*************************************************
c
c
        subroutine bactrk(termi,blink,iptrb,box,iptra,stackb,flag,time)
        integer termi,blink(50),iptrb,iptra,box(12,2),bacbox
        integer stackb(30),flag(12,2)
        double precision time
        if(termi.ge.17) then
             ia=blink(termi)
             iptrb = iptrb + 1
             time = time + 1.
             stackb(iptrb) = ia
             bacbox = (ia-7)/2
             if(ia/2*2.eq.ia) then
                  if(flag(bacbox,2).eq.0) box(bacbox,4)=0
                  if(flag(bacbox,2).eq.1) box(bacbox,3)=0
                  if(flag(bacbox,2).eq.2) box(bacbox,3) = 0
             endif
             if(ia/2*2.ne.ia) then
                  if(flag(bacbox,1).eq.0) box(bacbox,3)=0
                  if(flag(bacbox,1).eq.1) box(bacbox,4)=0
                  if(flag(bacbox,1).eq.2) box(bacbox,4)=0
             endif
        endif
        iptra = iptra - 1
        return
        end
c
c
c*********************************************************
c
c    Distributed Algorithm to control individual box
c
c*********************************************************
c
c
        subroutine protoc(sucpro,net)
c
c   D e c l a r a t i o n s
c
        integer ftop(50),fbot(50),btop(50),bbot(50),flink(50),
       *blink(50),proces(8),resour(8),flag(12,2),box(12,5),
       *a,b,stacka(30),stackb(30),curbox,pnum,rnum,sucpro
        double precision time
        character*5 net
        common ftop,fbot,btop,bbot,flink,blink,proces,resour,pnum,
       *rnum,flag,box,iptra,iptrb,time
c
```

```
c  I n i t i a l i z i n g
c
      time = 0.
      sucpro = 0
      iptra = pnum
      iptrb = 0
c
c  1 s t   s t a g e   p r o p a g a t i o n s
c
      do 10 i=1,pnum
      a = proces(pnum+1-i)
      if(net .eq. 'cube ') then
         if(a/2*2. ne. a) a=ftop(a)
         if(a/2*2. eq. a) a=fbot(a)
      endif
      if(net .eq. 'omega') then
         if(a. ge. 1 .and. a. le. 4) a=ftop(a)
         if(a. ge. 5 .and. a. le. 8) a=fbot(a)
      endif
      blink(a) = proces(pnum+1-i)
      time = time + 1.
   10 stacka(i) = a
c
      if(net .eq. 'omega') then
          do 11 jj=1,iptra
  11         stackb(jj) = stacka(jj)
          call sort(stackb,iptra)
          do 12 jj=1,iptra
  12         stacka(jj) = stackb(iptra+1-jj)
      endif
c
c  P r o p a g a t i o n s
c
   20 if(iptra.gt.0) then
        a=stacka(iptra)
        b=stacka(iptra-1)
        curbox = (a-7)/2
        if(a/2*2. ne. a .and. b. eq. a+1) then
          if(flag(curbox,1).ne.2 .and. flag(curbox,2).ne.2) then
             call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
             call bactrk(b,blink,iptrb,box,iptra,stackb,flag,time)
             go to 25
          endif
          if(box(curbox,3).ne.0 .and. box(curbox,4).ne.0) then
             call assign(0,ftop,fbot,flink,blink,a)
             call connec(1,1,0,a,sucpro,curbox,stackb)
          endif
          if(box(curbox,3).ne.0 .and. box(curbox,4).eq.0) then
             call assign(0,ftop,fbot,flink,blink,a)
             call connec(1,0,0,a,sucpro,curbox,stackb)
             call bactrk(b,blink,iptrb,box,iptra,stackb,flag,time)
          endif
          if(box(curbox,3).eq.0 .and. box(curbox,4).ne.0) then
             call assign(1,ftop,fbot,flink,blink,a)
             call connec(0,1,1,a,sucpro,curbox,stackb)
             call bactrk(b,blink,iptrb,box,iptra,stackb,flag,time)
          endif
          if(box(curbox,3).eq.0 .and. box(curbox,4).eq.0) then
             call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
             call bactrk(b,blink,iptrb,box,iptra,stackb,flag,time)
```

```
  endif
else
  if(a/2*2.ne.a) then
     if(flag(curbox,1).ne.2 .and. flag(curbox,2).ne.2) then
       call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
       go to 25
     endif
     if(box(curbox,3).ne.0) then
      if(flag(curbox,1).eq.2) then
        call assign(0,ftop,fbot,flink,blink,a)
        call connec(1,0,0,a,sucpro,curbox,stackb)
      else
        call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
        go to 25
      endif
     endif
     if(box(curbox,3).eq.0) then
      if(box(curbox,4).ne.0) then
       if(flag(curbox,2) .eq. 2) then
        call assign(1,ftop,fbot,flink,blink,a)
        call connec(0,1,1,a,sucpro,curbox,stackb)
       else
        call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
        go to 25
       endif
      endif
      if(box(curbox,4).eq.0) then
        call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
      endif
     endif
endif
if(a/2*2.eq.a) then
   if(flag(curbox,1).ne.2 .and. flag(curbox,2).ne.2) then
        call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
        go to 25
   endif
   if(box(curbox,3).ne.0) then
     if(flag(curbox,1) .eq. 2) then
       call assign(1,ftop,fbot,flink,blink,a-1)
       call connec(1,0,1,a,sucpro,curbox,stackb)
    else
       call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
       goto 25
    endif
   endif
   if(box(curbox,3).eq.0) then
     if(box(curbox,4).ne.0)then
      if(flag(curbox,2) .eq. 2) then
       call assign(0,ftop,fbot,flink,blink,a-1)
       call connec(0,1,0,a,sucpro,curbox,stackb)
      else
       call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
       goto 25
      endif
     endif
     if(box(curbox,4).eq.0) then
       call bactrk(a,blink,iptrb,box,iptra,stackb,flag,time)
     endif
   endif
endif
```

```
      endif
c
c   U p d a t e    s e r v i c e    s t a c k
c
   25 if(iptra.le.0 ) then
         iptra = iptrb
         call sort(stackb,iptrb)
         do 30 i=1,iptrb
   30    stacka(i) = stackb(iptrb+1-i)
         iptrb = 0
      endif
      go to 20
      endif
      return
      end
```

```
c*********************************************************************
c                                                                     *
c        D I S T R I B U T E D    C O N T R O L                       *
c                A L G O R I T H M                                    *
c                      I N                                            *
c        D Y N A M I C   E N V I R O N M E N T                        *
c                                                                     *
c  Program name   :  Dynamic.f                                        *
c  Language       :  Fortran 77                                       *
c  Programmer     :  Anthony Hicks                                    *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                         *
c                    Electrical Engineering Dept.                     *
c                    Purdue University                                *
c  Date written   :  02/27/82                                         *
c  Date updated   :  07/14/82                                         *
c                                                                     *
c  Description    :                                                   *
c      This program implement the Distributed Algorithm               *
c          to simulate the Cube & Omega interconnection               *
c          network model in a dynamic environment.                    *
c                                                                     *
c  Note :                                                             *
c      All "commented" write statements are used for debugging        *
c           purpose when modifying program is needed                  *
c                                                                     *
c  External Files :     Input  : Direct16.cub                         *
c                                Direct16.ohm                         *
c                                Junkin    System parameters          *
c                       Output : Statable  Statistics table           *
c                                                                     *
c*********************************************************************
c
c
c        **** M a i n   D r i v e r ****
c
c
c        D e c l a r a t i o n s
c
c  Numpro  : Number of processors
c  Pnum    : Number of requesting processors
c  Proces  : List of requesting processors
c  Pstate  : Processor's state: False = idle; True = request phase
c  Tlen    : Length of time to process that job on resources
c  Thold   : Length of time to hold resources
c  Pwait   : Time which this processor in wait state so far
c  Resreq  : Number of resources requested for current request
c  Ttrans  : Data transfer time
c  Transfl : Data transfer flag
c  Trscnt  : Data transfer time counter
c  Datime  : Preset allowed time to transfer data
c  Succnt  : # of requested resources actually made it so far/processor
c  Quitfl  : Flag to quit holding resources in request phase.
c  Reqcnt  : Total # of requests made so far per processor
c  Torsrq  : Total # of resources requested so far per processor
c  Probab  : A generated probability for general use
c  Pmark   : Decision mark for requesting certain # of resources
c  Prt     : Probability of requesting 1 processing unit time.
c  Sucreq  : Successful request count
c  Failrq  : Unsuccessful request count
c  Sucres  : Successful resources count
```

```
c  Failrs  : Unsuccessful resources count
c  Totlen  : Count of job length
c  Unluck  : # of resources made it for an unsuccessful request
c  Tdelay  : Total delay
c
c  Resour  : List of available output ports
c  Numsou  : Number of resources per output port
c  Rnum    : Number of free output ports
c  Freers  : Number of free resources per output port
c  Timere  : Time length to process jobs
c  Resor   : Contain which processor's work is being processed
c  Rstate  : Resource flag  0 : idle, 1 : busy.
c  Trunni  : Time spent on current job
c  Runfls  : Flag indicated processing in progress at that resource.
c
c  Avrsrq  : Average # of resources requested
c  Avrq1   : Running count of its square
c  Avsurs  : Average # of successful resources requested
c  Avsr1   : Running count of its square
c  Avflrs  : Average # of unsuccessful resources requested
c  Avfr1   : Running count of its square
c  Avlen   : Average length of time a job requested
c  Avlen1  : Running count of its square
c  Avdlay  : Average time delay for a resource to get thru the network
c  Avdly1  : Running count of its square
c  Avunlc  : Average count of unlucky resources requested
c  Avunl1  : Running count of its square
c  Totreq  : Total # of requests made in the system
c  Stopfl  : Stop flag to be used with blocking probability
c
c  Btrkfl  : Backtracking flag to indicate backtracking token
c  Dseed   : Double precision seed for distribution calls.
c  Dseed1  : Same as above but for another purpose.
c  Allres  : Total number of resources exixting on the network
c  Flink   : Forward link for a box terminal
c  Flink2  : Second forward link for broadcast
c  Blink   : Backward link from a box terminal
c  Stacka  : Event lists for forward propagation
c  Stackb  : Event lists for next cycle
c  Stackc  : Broadcast list for backward propagation
c  Stackd  : Broadcast list for next cycle
c  Stkare  : # of resources involved in this event (request)
c  Stkapr  : Which processor originated this request
c  Iptra   : Pointer to stack a
c  Iptrb   : Pointer to stack b
c  Iptrc   : Pointer to stack c
c  Tstart  : Time start of simulation, 0.
c  Tstop   : Length of simulation time
c  Net     : Type of network
c  Numsts  : Number of stages in the network
c  Busy    : Output terminal port flag  0 : free, 1 : busy.
c  Bstate   : Control box state 1 : both output ports available
c                               2 : both ports busy
c                               3 : top port free
c                               4 : bottom port free
c  Bmode    : Control box mode  0 : straight
c                               1 : diagonal
c                               2 : unassigned
c                               3 : lower broadcast
c                               4 : upper broadcast
```

```
c  Box     : Control box registers  Box(i,1) : Top terminal
c                                   Box(i,2) : Bottom terminal
c                                   Box(i,3) : # of available resources
c                                              as of here
c                                   Box(i,4) : # of      "      "
c                                   Box(i,5) : Total of 3 & 4 registers
c
c  Status  : Resource status at output port of this box
c                              1 : No resource available at either port
c                              2 : Top port has resources only
c                              3 : Only bottom port has resources
c                              4 : Equal # of resources at both ports
c                              5 : Top port has more resources.
c                              6 : Bottom one has more resources
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave,x,y,blocki,save
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numsts,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runflg(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),stopfl,trsflg(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numsts,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsflg,trscnt,datime,prt,store,
     *runflg,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
      common /count/ avrq1,avsr1,avfr1,avlen1,avdly1,avunl1
      data avrq1,avsr1,avfr1,avlen1,avdly1,avunl1/6*0./
c
      open(3,file='junkin',status='old',form='formatted')
      rewind 3
c
c     I n p u t   n e t w o r k   c o n n e c t i o n s
c
      read(5,2) net
    2 format(a5)
      do 7 i=1,96
    7 read(5,*) data(i),forwrd(i,1),forwrd(i,2),bckwrd(i,1),bckwrd(i,2)
c
c
   11 read(3,*) tstop,dseed,dseed1,numsou,thold,pmark,rtave,
     *xintvl,datime,numpro,numsts
      if(tstop .lt. 0) go to 99
      prt = 1./rtave
      tstart = 1
```

```
c
c      S t a r t   S i m u l a t i o n
c
       write(6,33) tstop,numsou,thold,pmark,rtave,xintvl,
      *datime,numpro,numsts
   33 format('Parameters of this simulation',//,
      *'      Length of simulation = ',i7,' units of time',/,
      *'      Number of resources/output port = ',i4,/,
      *'      Waiting time allowed/request    = ',i4,/,
      *'      Pmark       = ',f5.3,/,
      *'      Average resource time requested   = ',f7.3,/,
      *'      Blocking prob. quitting interval = ',f5.3,/,
      *'      Data transfer time = ',i4,/,
      *'      Case P = ',i4,/,
      *'      Number of stages in the network = ',i3)
c
c
c      I n i t i a l i z a t i o n s
c
       iptra = 0
       iptrb = 0
       iptrc = 0
       iptrd = 0
       rnum = numpro
       numbox = numsts*numpro/2
       allres = numpro * numsou
       stopfl = .false.
       kount = 0
       save = 0.1
c
       do 3 i=1,numpro
            do 4 j=1,numsou
               rstate(i,j) = 0
               trunni(i,j) = 0
               runfls(i,j) = .false.
    4          timere(i,j) = 0
            pstate(i) = .false.
            quitfl(i) = .false.
            freers(i) = numsou
            resour(i) = i
            tlen(i)   = 0
            reqcnt(i) = 0
            succnt(i) = 0
            torsrq(i) = 0
            pwait(i)  = 0
            totlen(i) = 0
            tdelay(i) = 0
            failrs(i) = 0
            failrq(i) = 0
            unluck(i) = 0
            sucreq(i) = 0
            sucres(i) = 0
    3       resreq(i) = 0
      do 5 i=1,400
         flink(i)  = 0
         flink2(i) = 0
         store(i) = 0
    5    btrkfl(i) = .false.
c
c     Set all boxes' terminal id's & clear all boxes' registers
```

```
c
        do 1 la=1,numbox
          busy(la,1) = .false.
          busy(la,2) = .false.
          bmode(la)  = 2
          bstate(la) = 1
          box(la,1) = la*2+15
    1 box(la,2) = box(la,1)+1
      do 9 i=1,numbox
          do 9 k=3,5
    9     box(i,k) = 0
c
c    B r o a d c a s t i n g
c
      call brocas
      do 19 j=1,numsts-1
         do 20 k=25,32
         jo = k-((j-1)*numpro/2)
         call broset(box,bckwrd,box(jo,1),box(jo,5),numpro)
c              call bxstat(jo)
   20    continue
   19 continue
c       do 44 jo=1,8
c  44 call bxstat(jo)
```

```
c*****************************************
   10 if(tstart.gt.tstop .or. stopfl) go to 98
c*****************************************
c
c
c     G e n e r a t i n g    A v a i l a b i l i t i e s
c
      call brocas
c     write(6,*) 'Brocas'
c
c
c     G e n e r a t i n g    R e q u e s t s
c
      call reqgen
c     write(6,*) 'Reqgen'
c
c
c     P r o c e s s   Broadcast   E v e n t s   L i s t
c
      call brobak(iptrc,iptrd,stackc,stackd,box,bckwrd,numpro)
c     write(6,*) 'Brobak'
c
c
c     P r o c e s s   Request   E v e n t s   L i s t
c
      call protoc
c     write(6,*) 'Protoc'
c
c
c     U p d a t e   C l o c k
c
      tstart = tstart + 1
      if(icount .eq. 1000) then
          icount = 1
          write(6,*) 'Time now is: ',tstart
      endif
c
c    Blocking probability check, if diffence interval
c      is very small, exit simulation
c
      kount = kount + 1
      if(kount .eq. 100) then
          kount = 0
          avflr2 = sumit(numpro,failrs)
          avsur2 = sumit(numpro,sucres)
          x      = avflr2 + avsur2
          if(x .eq. 0.) then
             x = 1.
          endif
          if(avflr2 .eq.0.) then
             blocki = 0.
          else
             blocki = avflr2/x
          endif
          if(abs(save-blocki).le.xintvl) stopfl = .true.
          save = blocki
      endif
      go to 10
c
c  Statistics calculation
```

```
c
   98 avrsr2 = sumit(numpro,torsrq)
      totreq = sumit(numpro,reqcnt)
      if(avrsr2 .eq. 0.) then
         avrsrq = 0.
      else
         avrsrq = avrsr2/totreq
      endif
c
      if(succt .lt. 1.)succt = 1.
      succt  = sumit(numpro,sucreq)
      avsur2 = sumit(numpro,sucres)
      if(avsur2 .eq. 0.) then
          avsurs = 0.
      else
          avsurs = avsur2/succt
      endif
c     write(6,*) 'I am here, stop 1'
c
      avle2  = sumit(numpro,totlen)
      if(avle2 .eq. 0.) then
          avlen = 0.
      else
          avlen  = avle2/totreq
      endif
c     write(6,*) 'I am here, stop 2'
c
      avdla2 = sumit(numpro,tdelay)
      if(avdla2 .eq. 0.) then
          avdlay = 0.
      else
          avdlay = avdla2/succt
      endif
c     write(6,*) 'I am here, stop 3'
c
      if(failct .lt. 1.)failct = 1.
      failct = sumit(numpro,failrq)
      avflr2 = sumit(numpro,failrs)
      if(avflr2 .eq. 0.) then
          avflrs = 0.
      else
          avflrs = avflr2/failct
      endif
c     write(6,*) 'I am here, stop 4'
c
      avunl2 = sumit(numpro,unluck)
      if(avunl2 .eq. 0.) then
          avunlc = 0.
      else
          avunlc = avunl2/failct
      endif
c     write(6,*) 'I am here, stop 5'
c
      y = avsur2 + avflr2
      if(y .lt. 1.) y=1.
      if(avflr2 .eq. 0.) then
         blocki = 0.
      else
         blocki = avflr2/y
      endif
```

```
      write(6,*) '     '
      write(6,*)  '   Collected Statistics of this run: '
      write(6,*) 'Quit at : ',tstart
      write(6,*) '     '
      write(6,12) avrsr2,totreq,avrsrq,avdlay,avlen
      write(6,14) avsur2,succt,avsurs,blocki,save
      write(6,15) avflr2,failct,avflrs,avunlc
      write(6,*) '     '
   12 format('Total # of resources requested = ',f10.2,/,
     *       'Total # of requests generated  = ',f10.2,/,
     *       'Average # of resources/request = ',f10.3,/,
     *       'Average time delay/request     = ',f10.3,/,
     *       'Average time length of a job   = ',f10.3)
   14 format('Total # of resources allocated = ',f10.2,/,
     *       'Total # of successful requests = ',f10.2,/,
     *       'Average # resources allocated/successful request = ',
     *f10.3,/,'Average blocking probability = ',f10.3,
     */,'Last computed blocking prob. = ',f10.3)
   15 format('Total # of unallocated resources = ',f10.2,/,
     *       'Total # of unsuccessful requests = ',f10.2,/,
     *       'Average # of resources/unsuccessful request = ',f10.3,/,
     *       'Average # of partially allocated resources/unsucc. req',
     *' = ',f10.3)
      pmark = pmark + pincrm
      go to 11
   99 stop
      end
```

```
c************************************************
c     Routine to Broadcast available resources
c************************************************
c
c
      subroutine brocas
c
      integer avail
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numsts,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runfls(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsfls(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numsts,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsfls,trscnt,datime,prt,store,
     *runfls,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
c  Update available resources registers
c
      do 9 i=1,numpro
          do 9 j=1,numsou
c
c       If this resource is active
c
               if(rstate(i,j).eq.1) then
c
c            For those have been in wait state
c
                    if(.not. runfls(i,j)) then
                        if(quitfl(resor(i,j))) then
                            freers(i) = freers(i) + 1
                            rstate(i,j) = 0
                        else
                        if(succnt(resor(i,j)).eq.resreq(resor(i,j)).and.
     *(.not. trsfls(resor(i,j)))) then
                            runfls(i,j) = .true.
                            trunni(i,j) = 1
                           endif
                        endif
c
c            For those have been running
c
                    else
                        trunni(i,j) = trunni(i,j) + 1
```

```
c
c           And  check for relinquish time
c
                  if(trunni(i,j).ge.timere(i,j)) then
                      freers(i)=freers(i)+1
                      rstate(i,j) = 0
                      runflg(i,j) = .false.
                  endif
                  endif
              endif
c                   call rstat(i,j)
    9 continue
c
c  Propagating backward one stage
c
      do 10 i=1,rnum
         avail = resour(i) + 80
         if(net .eq. 'cube ') then
            if(avail.ge.81 .and. avail.le.88) avail=bckwrd(avail,1)
            if(avail.ge.89 .and. avail.le.96) avail=bckwrd(avail,2)
         endif
         if(net .eq. 'omega') then
            if(avail/2*2 .ne. avail)avail = bckwrd(avail,1)
            if(avail/2*2 .eq. avail)avail = bckwrd(avail,2)
         endif
         ijk=(avail-numpro+1)/2
         if(avail.eq.box(ijk,1)) then
            new = freers(i) - box(ijk,3)
            box(ijk,3) = freers(i)
            box(ijk,5) = box(ijk,5) + new
         endif
         if(avail.eq.box(ijk,2)) then
            new = freers(i) - box(ijk,4)
            box(ijk,4) = freers(i)
            box(ijk,5) = box(ijk,5) + new
         endif
   10    continue
      return
      end
```

```
c*****************************************************
c  Routine to set boxes for Broadcasting uses
c*****************************************************
c
c
      subroutine broset(box,bckwrd,avail,sum,numpro)
c
      integer sum,bckwrd(400,2),link(2),box(64,5),curbox(2),avail
      integer numpro
c
      link(1) = bckwrd(avail,1)
      link(2) = bckwrd(avail,2)
      curbox(1) = (link(1)-numpro+1)/2
      curbox(2) = (link(2)-numpro+1)/2
   56 format(2x,4(2x,i5))
      do 11 j=1,2
         if(link(j).eq.box(curbox(j),1)) then
             new = sum - box(curbox(j),3)
             box(curbox(j),3) = sum
             box(curbox(j),5) = box(curbox(j),5) + new
         endif
         if(link(j).eq.box(curbox(j),2)) then
             new = sum - box(curbox(j),4)
             box(curbox(j),4) = sum
             box(curbox(j),5) = box(curbox(j),5) + new
         endif
   11 continue
      return
      end
```

```
c*****************************************************
c    Routine to set box output ports' & resources
c                      status
c*****************************************************
c
c
       subroutine boxset(busy,box,bstate,curbox,status)
c
       integer bstate(400),both,top,bottom,curbox,status(400)
       logical busy(400,2)
       integer box(64,5)
c
c    Box state setting
c
       top = 1
       bottom = 2
       if(.not.busy(curbox,top) .and. .not.busy(curbox,bottom)) then
            bstate(curbox) = 1
            endif
       if(busy(curbox,top).and.busy(curbox,bottom))bstate(curbox) = 2
       if((.not.busy(curbox,top)).and.busy(curbox,bottom)) then
            bstate(curbox) = 3
            endif
       if(busy(curbox,top).and.(.not.busy(curbox,bottom))) then
            bstate(curbox) = 4
            endif
c
c    Box Resource status setting
c
       top = 3
       bottom = 4
       both = 5
       if(box(curbox,top).gt.0.and.box(curbox,bottom).eq.0) then
            status(curbox) = 2
            endif
       if(box(curbox,top).eq.0.and.box(curbox,bottom).gt.0) then
            status(curbox) = 3
            endif
       if(box(curbox,top).eq.box(curbox,bottom))status(curbox)=4
       if(box(curbox,both).eq.0) status(curbox) = 1
       if(box(curbox,top).gt.0.and.box(curbox,bottom).gt.0) then
            if(box(curbox,top).gt.box(curbox,bottom))status(curbox)=5
            if(box(curbox,top).lt.box(curbox,bottom))status(curbox)=6
       endif
       return
       end
```

```
c*******************************************************
c   Routine to propagate broadcast availability
c          back stage by stage
c*******************************************************
c
c
      subroutine brobak(iptrc,iptrd,stackc,stackd,box,bckwrd,numpro)
c
      integer iptrc,iptrd,numpro,stackc(150),stackd(150)
      integer box(64,5),bckwrd(400,2),c,d
c
    3 if(iptrc.le.0) go to 11
      c = stackc(iptrc)
      if(c.gt.numpro/2) then
         d = c-numpro/2
         call broset(box,bckwrd,box(c,1),box(c,5),numpro)
c       call bxstat(c)
         iptrd = iptrd + 1
         stackd(iptrd) = d
      endif
      iptrc = iptrc - 1
      go to 3
   11 do 12 i=1,numpro/2
         iptrd = iptrd + 1
   12    stackd(iptrd) = i + 24
      iptrc = iptrd
      call sort(stackd,iptrd)
      do 20 i=1,iptrd
   20 stackc(i) = stackd(iptrd+1-i)
      iptrd = 0
      return
      end
```

```
c*********************************************************
c     R e q u e s t s   G e n e r a t i o n
c*********************************************************
c
c
      subroutine reqgen
c
      integer a,b
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numstg,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runflg(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsflg(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numstg,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsflg,trscnt,datime,prt,store,
     *runflg,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
      common /count/ avrq1,avsr1,avfr1,avlen1,avdly1,avunl1
c
      pnum = 0
      do 10 i=1,numpro
c
c     Generating new requests
c
          if (.not. pstate(i)) then
              call sgeot(dseed,1,pmark,xjunk,resreq(i))
              resreq(i) = resreq(i) -1
              resreq(i) = mod(resreq(i),allres)
              if(resreq(i) .gt. 0) then
                  call sgeot(dseed1,1,prt,xjunk,tlen(i))
c
c   Data transfer time can be modified here to be a variable
c     with certain distribution, now it is assumed to be
c          very fast.
c
                  ttrans(i) = datime
c
                  pstate(i) = .true.
                  trsflg(i) = .false.
                  pnum = pnum + 1
                  proces(pnum) = i
                  pwait(i) = 0
                  succnt(i) = 0
                  quitfl(i) = .false.
```

```
                totlen(i) = totlen(i) + tlen(i)
                avlen1 = avlen1 + tlen(i)**2
                reqcnt(i) = 1 + reqcnt(i)
                avrq1 = avrq1 + resreq(i)**2
                torsrq(i) = resreq(i) + torsrq(i)
             endif
          endif
c
c   For requesting processors in wait state
c
          if(pstate(i) .and. succnt(i) .lt. resreq(i)) then
c
c       Waiting period is up, abort request and relinquish
c           connections to free up the network
c
             if(pwait(i) .ge. thold) then
                totlen(i) = totlen(i) + tlen(i)
                pstate(i) = .false.
                failrq(i) = failrq(i) + 1
                failrs(i) = failrs(i) + resreq(i)
                avfr1 = avfr1 + resreq(i)**2
                unluck(i) = unluck(i) + succnt(i)
                avunl1 = avunl1 + succnt(i)**2
                quitfl(i) = .true.
               call freebx(i,net,busy,bmode,forwrd,flink,flink2,data,
     *btrkfl)
c               write(6,*) 'I am out of freebx'
             else
c
c      If waiting period is not up, updating waiting time
c
                pwait(i) = pwait(i) + 1
             endif
          endif
c
c   For requesting processors whose request is satisfied
c       Updating necessary statistics
c    and start data transfer from processor to resources.
c
          if(pstate(i).and. succnt(i) .ge. resreq(i).and.
     *trscnt(i).eq.0) then
             sucreq(i) = sucreq(i) + 1
             sucres(i) = sucres(i) + resreq(i)
             avrs1 = avrs1 + resreq(i)**2
             tdelay(i) = tdelay(i) + pwait(i)
             avdly1 = avdly1 + pwait(i)**2
          endif
c           call pstat(i)
c
c    I data transfer is completed, cut links and set all
c      exchange boxes involved free to partially free the network
c
          if(pstate(i).and.trsflg(i)) then
             trscnt(i) = trscnt(i) + 1
             if(trscnt(i) .ge. ttrans(i)) then
                pstate(i) = .false.
                call freebx(i,net,busy,bmode,forwrd,flink,flink2,data,
     *btrkfl)
c               write(6,*) 'I am out of freebx2'
                trsflg(i) = .false.
```

```
            endif
         endif
   10 continue
c
c  1 s t   s t a g e   p r o p a g a t i o n s
c
c     write(6,*) 'pnum = ',pnum,'  iptra = ',iptra
      do 15 i=1,pnum
      a = proces(pnum+1-i)
      b = a
      if(net .eq. 'cube ') then
         if(a/2*2.ne.a) a=forwrd(a,1)
         if(a/2*2.eq.a) a=forwrd(a,2)
      endif
      if(net .eq. 'omega') then
         if(a.ge.1 .and. a.le.numpro/2) a=forwrd(a,1)
         if(a.ge.numpro/2+1 .and. a.le.numpro) a=forwrd(a,2)
      endif
      blink(a) = b
      stkapr(a) = b
      stkare(a) = resreq(b)
      iptra = iptra + 1
      flink(a) = -1
   15 stacka(iptra) = a
c
         do 11 jj=1,iptra
   11       stackb(jj) = stacka(jj)
         call sort(stackb,iptra)
         do 12 jj=1,iptra
   12       stacka(jj) = stackb(iptra+1-jj)
      return
      end
```

```
c*********************************
c   Routine to list status of
c      System's components
c*********************************
c
      subroutine pstat(i)
c
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numsts,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runfls(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsfls(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numsts,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsfls,trscnt,datime,prt,store,
     *runfls,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
      write(6,1) i,pstate(i),resreq(i),tlen(i),quitfl(i),pwait(i),
     *succnt(i)
      write(6,2) reqcnt(i),torsrq(i),thold,pmark,prt
    1 format('P# = ',i2,'  Status = ',l1,'  # of resources rq = ',
     *i3,'  Job len = ',i4,'  Quit = ',l1,'  Wait = ',i3,'  Succnt = ',
     *i3)
    2 format('Request cnt = ',i4,'  Total Res. Rq = ',i4,
     *'  Time allow = ',i3,'  Pmark = ',f4.3,'  Tmark = ',f4.3)
      return
c
      entry rstat(i,j)
      write(6,3) i,j,rstate(i,j),runfls(i,j),resor(i,j),trunni(i,j)
      write(6,4) timere(i,j),freers(i),tstart
    3 format('R# = ',i2,',',i2,'  Status = ',l1,'  Runfls = ',l1,
     *'  Orig. Proc = ',i2,'  Time ran = ',i4)
    4 format('  Time rq = ',i4,'  Free rs. = ',i3,'  Time now = ',i5)
      return
c
      entry bxstat(i)
      write(6,5) i,(box(i,j),j=1,5)
      write(6,6) bmode(i),bstate(i),(busy(i,jj),jj=1,2),status(i)
    5 format('Box # = ',i2,'  T = ',i2,'  B = ',i2,'  T.Res = ',i3,
     *'  B.Res = ',i3,'  Total = ',i3)
    6 format('Setting = ',i1,'  Outport stat = ',i1,' T.busy = ',l1,
     *'  B.busy = ',l1,'  Reso stat = ',i1)
      return
c
      end
```

```
c*******************************************************
c   Routine to relinquish communication lines
c           and free exchange boxes
c*******************************************************
c
      subroutine freebx(proc,net,busy,bmode,forwrd,flink,flink2,data,
     *btrkfl)
c
      integer proc,nupro,curbox,bmode(400),data(400),flink(400)
      integer stack(150),flink2(400),forwrd(400,2),k
      logical busy(400,2),btrkfl(400)
      character*5 net
c
c     write(6,*) 'I am in freebx'
      top = 1
      bottom = 2
      nupro = proc
      if(net .eq.'cube ') then
          if(nupro/2*2.ne.nupro)nupro=forwrd(nupro,1)
          if(nupro/2*2.eq.nupro)nupro=forwrd(nupro,2)
      endif
      if(net .eq.'omega') then
          if(nupro.ge.1 .and. nupro.le.numpro/2)nupro=forwrd(nupro,1)
      if(nupro.ge.numpro/2+1.and.nupro.le.numpro)nupro=forwrd(nupro,2)
      endif
c
      k = 0
c
c  Flink is used here to detect if this token has made connection
c    in present box.
c
   10 if(data(nupro) .gt. 0 .or. flink(nupro).lt.0) go to 20
          curbox = (nupro-15)/2
c         write(6,*) 'nupro = ',nupro,' curbox = ',curbox,
c    * 'bmode = ',bmode(curbox),' flink = ',flink(nupro),
c    * 'flink2 = ',flink2(nupro)
          if(bmode(curbox) .eq. 0) then
              if(nupro/2*2 .ne. nupro)busy(curbox,top) = .false.
              if(nupro/2*2 .eq. nupro)busy(curbox,bottom) = .false.
          endif
          if(bmode(curbox) .eq. 1) then
              if(nupro/2*2 .ne. nupro)busy(curbox,bottom)=.false.
              if(nupro/2*2 .eq. nupro)busy(curbox,top)=.false.
          endif
          if(bmode(curbox) .eq. 3 .or. bmode(curbox) .eq. 4) then
              k = k + 1
              stack(k) = flink2(nupro)
              busy(curbox,top) = .false.
              busy(curbox,bottom) = .false.
              bmode(curbox) = 2
          endif
          btrkfl(nupro) = .false.
          nupro = flink(nupro)
          go to 10
c
c Flink2 is used here to mark off a traveling token which
c   needs not to be on stack anymore because its requesting
c   processor just relinquish the network
c
   20 flink2(nupro) = -2
```

```
if(k .le. 0) return
numro = stack(k)
k = k - 1
go to 10
end
```

```
c*********************************************************
c
c    Distributed Algorithm to control individual box
c
c*********************************************************
c
c
      subroutine protoc
c
      integer a,b,c,remain,type1,type2,type3,curbox,seed
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numstg,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runflg(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsflg(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numstg,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsflg,trscnt,datime,prt,store,
     *runflg,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
c  P r o p a g a t i o n s
c
          curbox = 64
        call boxset(busy,box,bstate,curbox,status)
      seed = int(dseed1)
  900 if(iptra.le.0) go to 600
        a=stacka(iptra)
        b=stacka(iptra-1)
c     write(6,*) 'Last box setting'
c     write(6,5) curbox,(box(curbox,j),j=1,5)
c     write(6,6) bmode(curbox),bstate(curbox),(busy(curbox,jj),jj=1,2),
c    * status(curbox)
c   5 format('Box # = ',i2,'  T = ',i2,'  B = ',i2,'  T.Res = ',i3,
c    *'  B.Res = ',i3,'  Total = ',i3)
c   6 format('Setting = ',i1,'  Outport stat = ',i1,' T.busy = ',l1,
c    *'  B.busy = ',l1,'  Reso stat = ',i1)
c     write(6,*) ' '
c     write(6,999) a,b,iptra,iptrb
c     write(6,*) (stacka(ll),ll=1,iptra)
c     write(6,*) (stackb(ll),ll=1,iptrb)
c     write(6,998) stkare(a),stkapr(a),stkare(b),stkapr(b)
c 998  format('top res & from = ',i2,i3,' Bot res & from = ',i2,i3)
c 999  format('top input = ',i2,'  Bottom input = ',i2,' iptra = ',
c    * i3,' iptrb = ',i3)
c
```

```
c  Here a relinquished processor's token is screened out.
c
      iptra = iptra - 1
      if(flink2(a) .lt. 0) then
          flink2(a) = 0
          go to 900
      endif
      curbox = (a-15)/2
      call boxset(busy,box,bstate,curbox,status)
c         call bxstat(curbox)
c
c   Case of two signals arriving at one box at the same time
c
      if(a/2*2.ne.a .and. b.eq.a+1) then
          iptra = iptra - 1
c     write(6,*) 'Going to 2'
          go to 2
        else
c
c   Case of only 1 signal arriving at top input
c
          if(a/2*2.ne.a) then
              type1 = 0
              type2 = 1
              type3 = 4
              c     = a
c     write(6,*) 'Going to 1 from top'
              go to 1
          endif
c
c   Case of only 1 signal entering at bottom input
c
          if(a/2*2.eq.a) then
              type1 = 1
              type2 = 0
              type3 = 3
              c     = a-1
c     write(6,*) 'Going to 1 from bottom'
              go to 1
          endif
        endif
c
c
c  NOTE: See declarations for meanings of 'Status'
c
c
    1 go to(21,22,23,24),bstate(curbox)
c
c      Both output ports available (not busy)
c
   21 go to(22,23,24,25,26,27),status(curbox)
c
   25 if(stkare(a) .le. box(curbox,3)) go to 23
   29 if(stkare(a) .gt. box(curbox,5)) then
          remain = stkare(a) - box(curbox,5)
          call bactrk(a,remain)
          stkar2(a) = box(curbox,4)
      endif
      if(stkare(a) .le. box(curbox,5))stkar2(a)=stkare(a)-box(curbox,3)
      stkare(a) = box(curbox,3)
```

```
      call assign(type3,bmode(curbox),forwrd,flink,flink2,blink,c)
      call connec(1,1,type3,a,curbox)
c     write(6,*) 'I was at 1-25'
      go to 900
c
   26 if(stkare(a) .le. box(curbox,4)) go to 24
      if(stkare(a) .le. box(curbox,3)) go to 23
c     write(6,*) 'I was at 1-26'
      go to 29
c
   27 if(stkare(a) .le. box(curbox,3)) go to 23
      if(stkare(a) .le. box(curbox,4)) go to 24
      if(stkare(a) .gt. box(curbox,5)) then
          remain = stkare(a) - box(curbox,5)
          call bactrk(a,remain)
          stkar2(a) = box(curbox,3)
      endif
      if(stkare(a) .le. box(curbox,5))stkar2(a)=stkare(a)-box(curbox,4)
      stkare(a) = box(curbox,4)
      call assign(type3,bmode(curbox),forwrd,flink,flink2,blink,c)
      call connec(1,1,type3,a,curbox)
c     write(6,*) 'I was at 1-27'
      go to 900
c
c      Both output ports busy (unavailable)
c
   22 call bactrk(a,stkare(a))
c     write(6,*) 'I was at 1-22'
      go to 900
c
c      Only top port is free (not busy)
c
   23 if(box(curbox,3).ne. 0) then
          if(stkare(a).gt.box(curbox,3)) then
              remain = stkare(a) - box(curbox,3)
              stkare(a) = box(curbox,3)
              call bactrk(a,remain)
          endif
          call assign(type1,bmode(curbox),forwrd,flink,flink2,blink,c)
          call connec(1,0,type1,a,curbox)
      else
          call bactrk(a,stkare(a))
      endif
c     write(6,*) 'I was at 1-23'
      go to 900
c
c      Only bottom output port available (not busy)
c
   24 if(box(curbox,4).ne. 0) then
          if(stkare(a) .gt. box(curbox,4)) then
              remain = stkare(a) - box(curbox,4)
              stkare(a) = box(curbox,4)
              call bactrk(a,remain)
          endif
          call assign(type2,bmode(curbox),forwrd,flink,flink2,blink,c)
          call connec(0,1,type2,a,curbox)
      else
          call bactrk(a,stkare(a))
      endif
c     write(6,*) 'I was at 1-24'
```

```
      go to 900
c
c
c
c
    2 go to(200,100,300,400),bstate(curbox)
c
c      Both output ports are busy
c
  100 call bactrk(a,stkare(a))
      call bactrk(a+1,stkare(a+1))
c     write(6,*) 'I was at 2-100'
      go to 900
c
c      Both output ports are not busy (available)
c
  200 if(stkare(a)-stkare(a+1))210,220,230
c
  210 go to(100,310,410,234,236,234),status(curbox)
c
  220 go to(100,222,223,234,234,234),status(curbox)
  222 chance = ranf(seed)
      if(chance .gt. .5)go to 330
c     write(6,*) 'I was at 2-222'
      go to 310
  223 chance = ranf(seed)
      if(chance .gt. .5)go to 430
c     write(6,*) 'I was at 2-223'
      go to 410
c
  230 go to(100,330,430,234,234,236),status(curbox)
  234 call go1(a,curbox,stkare,box)
      call go2(a+1,curbox,stkare,box)
      call assign(0,bmode(curbox),forwrd,flink,flink2,blink,a)
      call connec(1,1,0,a,curbox)
c     write(6,*) 'I was at 2-234'
      go to 900
  236 call go2(a,curbox,stkare,box)
      call go1(a+1,curbox,stkare,box)
      call assign(1,bmode(curbox),forwrd,flink,flink2,blink,a)
      call connec(1,1,1,a,curbox)
c     write(6,*) 'I was at 2-236'
      go to 900
c
c      Only top output port is free
c
  300 if(stkare(a) - stkare(a+1))310,320,330
c
  310 call bactrk(a,stkare(a))
      type1 = 1
      a = a+1
      c = a-1
c     write(6,*) 'I was at 2-310'
      go to 23
c
  320 chance = ranf(seed)
      if(chance .gt. .5)go to 310
c     write(6,*) 'I was at 2-320'
      go to 330
c
```

```
  330 call bactrk(a+1,stkare(a+1))
      type1 = 0
      c = a
c     write(6,*) 'I was at 2-330'
      go to 23
c
c      Only bottom port is free
c
  400 if(stkare(a)-stkare(a+1))410,420,430
c
  410 call bactrk(a,stkare(a))
      type2 = 0
      a = a+1
      c = a-1
c     write(6,*) 'I was at 2-410'
      go to 24
c
  420 chance = ranf(seed)
      if(chance .gt. .5)go to 410
c     write(6,*) 'I was at 2-420'
      go to 430
c
  430 call bactrk(a+1,stkare(a+1))
      type2 = 1
      c = a
c     write(6,*) 'I was at 2-430'
      go to 24
c
c   U p d a t e    s e r v i c e    s t a c k
c
  600 iptra = iptrb
      call sort(stackb,iptrb)
      do 130 i=1,iptrb
  130    stacka(i) = stackb(iptrb+1-i)
      iptrb = 0
      return
      end
```

```
c*********************************
c  Routines to assign and connec
c*********************************
c
c
      subroutine go1(a,curbox,stkare,box)
c
      inteser a,curbox,stkare(400),remain,box(64,5)
c
c     write(6,*) 'I was in Go1'
          if(stkare(a).gt.box(curbox,3)) then
              remain = stkare(a) - box(curbox,3)
              stkare(a) = box(curbox,3)
              call bactrk(a,remain)
          endif
          return
          end
c
c
c
      subroutine go2(a,curbox,stkare,box)
c
      inteser a,curbox,stkare(400),remain,box(64,5)
c
c     write(6,*) 'I was in Go2'
          if(stkare(a) .gt. box(curbox,4)) then
              remain = stkare(a) - box(curbox,4)
              stkare(a) = box(curbox,4)
              call bactrk(a,remain)
          endif
          return
          end
```

```
c*******************************************************
c  Routine to update succesful processors counter
c*******************************************************
c
      subroutine update(termi,linky,stk1)
c
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer linky(400),termi,stk1(400)
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numstg,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runflg(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsflg(400)
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numstg,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsflg,trscnt,datime,prt,store,
     *runflg,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
c     write(6,*) 'I was in update'
      if(termi.ge.65 .and. termi.le.80) then
        if(linky(termi).ge.81 .and. linky(termi).le.96) then
c     write(6,*) 'I made it from proc. ',stkapr(termi)
            k = 0
            kk = linky(termi)-80
            do 10 i=1,numsou
                if(rstate(kk-80,i) .ne. 1) then
                    resor(kk,i) = stkapr(termi)
                    trunni(kk,i) = 0
                    rstate(kk,i) = 1
                    runflg(kk,i) = .false.
                    timere(kk,i) = tlen(resor(kk,i))
                    freers(kk) = freers(kk) - 1
                    succnt(stkapr(termi)) = succnt(stkapr(termi)) + 1
                    if(succnt(stkapr(termi)).ge.resreq(stkapr(termi)))
     * then
                        trsflg(stkapr(termi)) = .true.
                        trscnt(stkapr(termi)) = 0
                    endif
                    k = k + 1
                endif
                if(k.ge.stk1(termi))  return
   10       continue
         endif
       else
           iptrb = iptrb+1
           flink(linky(termi)) = -1
           stackb(iptrb) = linky(termi)
```

```
            stk1(linky(termi)) = stk1(termi)
            stkapr(linky(termi)) = stkapr(termi)
         endif
         return
         end
c
c
c*********************************
c   Routine to sum arrays
c*********************************
c
c
         real function sumit(numpro,array1)
         integer numpro
         integer array1(numpro)
         sumit = 0.
         do 10 i=1,numpro
      10 sumit = sumit + float(array1(i))
         return
         end
c
c
c*****************
c  Sort Routine
c*****************
c
c
         subroutine sort(x,n)
         integer x(n)
         do 10 m=2,n
         k = m
       5 if(x(k) .le. x(k/2))go to 10
         ky= x(k)
         x(k) = x(k/2)
         x(k/2) = ky
         k= k/2
         if(k.gt.1) go to 5
      10 continue
         do 30 nm=2,n
         m = n-nm + 2
         itop = x(1)
         j = 1
      25 if(m.le.j*2) go to 27
         if(x(2*j).ge.x(2*j+1))i=2*j
         if(x(2*j).lt.x(2*j+1))i=2*j+1
         if(x(i).le.x(m))go to 27
         x(j) = x(i)
         j=i
         go to 25
      27 x(j) = x(m)
      30 x(m) = itop
         return
         end
```

```
c*********************************
c  Routine to make connections
c*********************************
c
c
      subroutine connec(t,b,mode,termi,curbox)
c
      integer top,bottom,t,b,curbox,termi,mode
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numsts,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runfls(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsfls(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numsts,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsfls,trscnt,datime,prt,store,
     *runfls,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
c     write(6,*) 'I was in connec'
      top = 1
      bottom = 2
      moda = mode + 1
      go to(10,10,25,30,30), moda
   10 if(t.eq.1) busy(curbox,top) = .true.
      if(b.eq.1) busy(curbox,bottom) = .true.
c
c  If a backtracked signal is making a connection, its box mode
c     needs to be changed to 'broadcast' if it is a partial request
c
      if(btrkfl(termi)) then
          btrkfl(termi) = .false.
          if(stkare(termi) .lt. store(termi)) then
             if(termi/2*2 .ne. termi) then
                 call assign(4,bmode(curbox),forwrd,flink,flink2,blink,
     *termi)
                 if(mode .eq. 1) then
                     stkar2(termi) = stkare(termi)
                     go to 31
                 endif
                 if(mode .eq. 0) go to 32
             endif
             if(termi/2*2 .eq. termi) then
                 call assign(3,bmode(curbox),forwrd,flink,flink2,blink,
     *termi-1)
                 if(mode .eq. 0) then
```

```
                stkar2(termi) = stkare(termi)
                go to 31
              endif
              if(mode .eq. 1) go to 32
            endif
          endif
        endif
32 call update(termi,flink,stkare)
   if(t .eq.1 .and. b .eq. 1)call update(termi+1,flink,stkare)
   return
25 busy(curbox,top) = .false.
   busy(curbox,bottom) = .false.
   return
30 busy(curbox,top) = .true.
   busy(curbox,bottom) = .true.
   call update(termi,flink,stkare)
31 call update(termi,flink2,stkar2)
   return
   end
```

```
                  box(bacbox,4) = 0
               endif
               flink(ia) = -1
            endif
         else
            btrkfl(ia) = .true.
            iptrb = iptrb + 1
            stackb(iptrb) = ia
            flink(ia) = -1
            store(ia) = stkare(ia)
            stkare(ia) = it
            stkarr(ia) = stkarr(termi)
            if(bmode(bacbox).eq.0) then
               if(ia/2*2 .ne. ia) then
                  if(stkare(ia).eq.resreq(stkarr(ia))) then
                     busy(bacbox,1) = .false.
                  endif
                  box(bacbox,5) = box(bacbox,4)
                  box(bacbox,3) = 0
               else
                  if(stkare(ia).eq.resreq(stkarr(ia))) then
                     busy(bacbox,2) = .false.
                  endif
                  box(bacbox,5) = box(bacbox,3)
                  box(bacbox,4) = 0
               endif
            endif
            if(bmode(bacbox).eq.1) then
               if(ia/2*2 .ne. ia) then
                  if(stkare(ia).eq.resreq(stkarr(ia))) then
                     busy(bacbox,2) = .false.
                  endif
                  box(bacbox,5) = box(bacbox,3)
                  box(bacbox,4) = 0
               else
                  if(stkare(ia).eq.resreq(stkarr(ia))) then
                     busy(bacbox,1) = .false.
                  endif
                  box(bacbox,5) = box(bacbox,4)
                  box(bacbox,3) = 0
               endif
            endif
         endif
      endif
c
c  Case of rejected signals are at 1st stage:
c       Put them on stack and try propagating forward again
c
      if(termi .lt. numpro*2+1) then
         iptrb = iptrb + 1
         flink(termi) = -1
         stackb(iptrb) = termi
      endif
      return
      end
```

```
c*********************************************
c  Routine to backtrack whenever required
c*********************************************
c
c
      subroutine bactrk(termi,it)
c
      integer termi,bacbox
      character*5 net
      double precision dseed,dseed1
      real pmark,rtave
      integer sucreq(64),sucres(64),failrq(64),failrs(64),tdelay(64)
      integer totlen(64),unluck(64),stkar2(400),ttrans(64)
      integer flink2(400),status(400),stacka(400),stackb(400)
      integer tstart,tstop,freers(64),numsou,succnt(400),store(400)
      integer thold,allres,numpro,tlen(64),resreq(64),trscnt(64),datime
      integer resor(64,10),timere(64,10),rstate(64,10),trunni(64,10)
      integer box(64,5),pwait(64),stackc(150),stackd(150),stkare(400)
      integer numsts,reqcnt(64),bstate(400),stkapr(400),torsrq(64)
      integer data(400),bmode(400),bckwrd(400,2),flink(400),blink(400)
      integer proces(64),resour(64),sucpro,pnum,rnum,forwrd(400,2)
      logical runflg(64,10),pstate(64),quitfl(64),busy(400,2)
      logical btrkfl(400),trsflg(400)
c
      common sucreq,sucres,failrq,failrs,tdelay,unluck,stkar2,
     *flink,flink2,blink,ttrans,stacka,stackb,totlen,thold,allres,
     *status,tstart,tstop,dseed,dseed1,freers,numsou,succnt,numpro,
     *tlen,resreq,resor,timere,rstate,trunni,box,sucpro,busy,pmark,
     *pwait,stackc,stackd,stkare,stkapr,numsts,reqcnt,bstate,rtave,
     *torsrq,data,bmode,forwrd,bckwrd,proces,resour,pnum,rnum,btrkfl,
     *trsflg,trscnt,datime,prt,store,
     *runflg,pstate,quitfl,iptra,iptrb,iptrc,iptrd,net
c
c Case of rejected signals not at 1st stage
c
c     write(6,*) 'I was in bactrk'
      if(termi.ge.numpro*2+1) then
          ia=blink(termi)
          bacbox = (ia-15)/2
          if(btrkfl(termi)) btrkfl(termi)= .false.
c
c   Checking if it is the 2nd backtracking signal to merge 'em if
c      necessary, set RA's accordingly, and clear busy flag for
c      output port if it is not a partial backtrack.
c
          if(btrkfl(ia)) then
            if(bmode(bacbox).eq.3 .or. bmode(bacbox).eq.4) then
               stkare(ia) = stkare(ia) + it
               if(termi.eq.flink(ia)) then
                  if(stkare(ia).eq.resreq(stkapr(ia))) then
                      busy(bacbox,1) = .false.
                  endif
                  box(bacbox,5) = box(bacbox,4)
                  box(bacbox,3) = 0
               endif
               if(termi.eq.flink2(ia)) then
                  if(stkar2(ia).eq.resreq(stkapr(ia))) then
                      busy(bacbox,2) = .false.
                  endif
                  box(bacbox,5) = box(bacbox,3)
```

```
c**********************************
c  Box function set routine
c**********************************
c
c
      subroutine assign(mode,numode,port,flink,flink2,blink,index)
c
      integer port(400,2), blink(400), flink(400)
      integer flink2(400),top,bottom,numode,mode
c
c     write(6,*) 'I was in Assign'
      top = 1
      bottom = 2
      numode = mode
      moda = mode + 1
      go to (10,20,50,30,40),moda
c
c             Straight   Mode = 0
c
   10 flink(index) = port(index,top)
      flink(index+1) = port(index+1,bottom)
      blink(port(index,top)) = index
      blink(port(index+1,bottom)) = index + 1
      return
c
c            Diagonal   Mode = 1
c
   20 flink(index) = port(index,bottom)
      flink(index+1) = port(index+1,top)
      blink(port(index,bottom)) = index
      blink(port(index+1,top)) = index + 1
      return
c
c        Lower Broadcast    Mode = 3
c
   30 flink(index+1) = port(index+1,top)
      flink2(index+1) = port(index+1,bottom)
      blink(port(index+1,top)) = index+1
      blink(port(index+1,bottom)) = index+1
      return
c
c       Upper Broadcast    Mode = 4
c
   40 flink(index) = port(index,top)
      flink2(index) = port(index,bottom)
      blink(port(index,top)) = index
      blink(port(index,bottom)) = index
   50 return
      end
```

```
c*****************************************************************
c                                                                 *
c     S t a t i s t i c s    C a l c u l a t i o n s               *
c                                                                 *
c  Program name   :  Calstat.f                                    *
c  Language       :  Fortran 77                                   *
c  Programmer     :  Anthony Hicks                                *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                     *
c                    Electrical Engineering Dept.                 *
c                    Purdue University                            *
c  Date written   :  10/20/81                                     *
c  Date updated   :  10/20/81                                     *
c                                                                 *
c  Description    :                                               *
c      Blocking information calculations                          *
c                                                                 *
c  External Files  -  Input   : Statable  Statistics table        *
c                     Output  : Statinfo  Statistics              *
c                                                                 *
c*****************************************************************
c
c
      real varian(8,8),stdev(8,8),pmean(8,8),blkpro(8,8),
     *palloc(8,8),datab(8,8,3),rowtot(8)
      data rowtot,total/9*0./
      do 10  k=1,3
          do 10 i=1,8
   10     read(5,*) (datab(i,j,k),j=1,8)
      do 20 i=1,8
          do 20 j=1,8
               pmean(i,j)  = datab(i,j,1)/datab(i,j,2)
               palloc(i,j) = pmean(i,j)/float(i)
               blkpro(i,j) = 1. - palloc(i,j)
               varian(i,j) = datab(i,j,3)/datab(i,j,2)-pmean(i,j)**2
               stdev(i,j)  = sqrt(varian(i,j))
               rowtot(i)   = rowtot(i) + datab(i,j,2)
   20          total       = total + datab(i,j,2)
      write(6,21)
   21 format(4x,'resource:  ',3x,'1',11x,'2',11x,'3',11x,'4',11x,'5',
     *11x,'6',11x,'7',11x,'8',/,2x,'processor')
      write(6,31)
      do 30 i=1,8
   30    write(6,32) i,(pmean(i,j),j=1,8)
      write(6,41)
      do 40 i=1,8
   40    write(6,32) i,(varian(i,j),j=1,8)
      write(6,51)
      do 50 i=1,8
   50    write(6,32) i,(stdev(i,j),j=1,8)
      write(6,61)
      do 60 i=1,8
   60    write(6,32) i,(palloc(i,j),j=1,8)
      write(6,71)
      do 70 i = 1,8
   70    write(6,32) i,(blkpro(i,j),j=1,8),rowtot(i)
      write(6,73) total
   31 format(/,20x,'average # of processors allocated',/)
   41 format(/,20x,'variance of processors allocated',/)
   32 format(5x,i1,4x,8(f10.5,2x),4x,f6.0,' cases')
   51 format(/,20x,'standard deviation of processors allocations',/)
```

```
61 format(/,20x,'average percentage of processors allocated',/)
71 format(/,20x,'average processor blocking probabilities',/)
73 format(//,60x,'total = ',f6.0,' cases')
   stop
   end
```

```
c*********************************************************************
c                                                                     *
c     A v e r a g e   D e l a y s   C a l c u l a t i o n s           *
c                                                                     *
c  Program name   :  Caltime.f                                       *
c  Language       :  Fortran 77                                      *
c  Programmer     :  Anthony Hicks                                   *
c  Facilities     :  UNIX Dual-Vax 780 on ECN                        *
c                    Electrical Engineering Dept.                   *
c                    Purdue University                              *
c  Date written   :  01/13/82                                        *
c  Date updated   :  01/13/82                                        *
c                                                                     *
c  Description    :                                                  *
c      Average delaying time calculations.                          *
c                                                                     *
c  External Files  -  Input   : Statable  Statistics table           *
c                     Output  : Stattime  Average delays            *
c                                                                     *
c*********************************************************************
c
c
      real*8 varian(8,8),stdev(8,8),pmean(8,8),blkpro(8,8),
     *palloc(8,8),datab(8,8,3),rowtot(8)
      data rowtot,total/9*0./
      do 10  k=1,3
          do 10 i=1,8
   10     read(5,*) (datab(i,j,k),j=1,8)
      do 20 i=1,8
          do 20 j=1,8
               pmean(i,j)  = datab(i,j,1)/datab(i,j,2)
               varian(i,j) = datab(i,j,3)/datab(i,j,2)-pmean(i,j)**2
               stdev(i,j)  = sqrt(varian(i,j))
               rowtot(i)   = rowtot(i) + datab(i,j,2)
   20          total       = total + datab(i,j,2)
      write(6,21)
   21 format(4x,'resource:  ',3x,'1',11x,'2',11x,'3',11x,'4',11x,'5',
     *11x,'6',11x,'7',11x,'8',/,2x,'processor')
      write(6,31)
      do 30 i=1,8
   30    write(6,32) i,(pmean(i,j),j=1,8)
      write(6,41)
      do 40 i=1,8
   40    write(6,32) i,(varian(i,j),j=1,8)
      write(6,51)
      do 50 i=1,8
   50    write(6,32) i,(stdev(i,j),j=1,8)
      write(6,73) total
   31 format(/,20x,'average units of delaying time',/)
   41 format(/,20x,'variance of delaying time',/)
   32 format(5x,i1,4x,8(f10.5,2x),f6.0)
   51 format(/,20x,'standard deviation of delaying time',/)
   73 format(//,60x,'total = ',f6.0,' cases')
      stop
      end
```

```
Mar 16 17:05 1982    COMMAND FILE: RUNALL    Page 1

createree  <1235.4678>  12354678
cp    123546.78    procreq
findit      <12354678
cp statable table.bak
rm    12354678
createree  <1236.4578>  12364578
cp    123645.78    procreq
findit      <12364578
cp statable table.bak
rm    12364578
createree  <1237.4568>  12374568
cp    123745.68    procreq
findit      <12374568
cp statable table.bak
rm    12374568
createree  <1238.4567>  12384567
cp    123845.67    procreq
findit      <12384567
cp statable table.bak
rm    12384567
createree  <1234.5678>  12345678
cp    123456.78    procreq
findit      <12345678
cp statable table.bak
rm    12345678
createree  <8356.2471>  83562471
cp    835624.71    procreq
findit      <83562471
cp statable table.bak
rm    83562471
createree  <8567.2413>  85672413
cp    856724.13    procreq
findit      <85672413
cp    statable table.bak
rm    85672413
createree  <8543.2617>  85432617
cp    854326.17    procreq
findit      <85432617
cp    statable table.bak
rm    85432617
createree  <2857.6314>  28576314
cp    285763.14    procreq
findit      <28576314
cp statable table.bak
rm    28576314
createree  <2734.6815>  27346815
cp    273468.15    procreq
findit      <27346815
cp statable table.bak
rm    27346815
createree  <8756.4312>  87564312
cp    875643.12    procreq
findit      <87564312
cp statable table.bak
rm    87564312
createree  <7543.2816>  75432816
cp    754328.16    procreq
findit      <75432816
cp statable table.bak
rm    75432816
createree  <1245.3678>  12453678
cp    124536.78    procreq
findit      <12453678
cp statable table.bak
rm    12453678
createree  <1246.3578>  12463578
cp    124635.78    procreq
findit      <12463578
cp statable table.bak
rm    12463578
createree  <1247.3568>  12473568
cp    124735.68    procreq
findit      <12473568
cp statable table.bak
rm    12473568
createree  <1248.3567>  12483567
cp    124835.67    procreq
findit      <12483567
cp statable table.bak
rm    12483567
createree  <1256.3478>  12563478
cp    125634.78    procreq
findit      <12563478
cp statable table.bak
rm    12563478
createree  <1257.3468>  12573468
cp    125734.68    procreq
findit      <12573468
cp statable table.bak
rm    12573468
createree  <1258.3467>  12583467
cp    125834.67    procreq
findit      <12583467
cp statable table.bak
rm    12583467
createree  <1267.3458>  12673458
cp    126734.58    procreq
findit      <12673458
cp statable table.bak
rm    12673458
createree  <1268.3457>  12683457
cp    126834.57    procreq
findit      <12683457
cp statable table.bak
rm    12683457
createree  <1278.3456>  12783456
cp    127834.56    procreq
findit      <12783456
cp statable table.bak
rm    12783456
createree  <1345.2678>  13452678
cp    134526.78    procreq
```

```
Mar 16 17:05 1982  runall Page 2

findit      <13452678
cp statable table.bak
rm    13452678
creatree    <1346.2578> 13462578
cp    134625.78    procreq
findit      <13462578
cp statable table.bak
rm    13462578
creatree    <1347.2568> 13472568
cp    134725.68    procreq
findit      <13472568
cp statable table.bak
rm    13472568
creatree    <1348.2567> 13482567
cp    134825.67    procreq
findit      <13482567
cp statable table.bak
rm    13482567
creatree    <1356.2478> 13562478
cp    135624.78    procreq
findit      <13562478
cp statable table.bak
rm    13562478
creatree    <1357.2468> 13572468
cp    135724.68    procreq
findit      <13572468
cp statable table.bak
rm    13572468
creatree    <1358.2467> 13582467
cp    135824.67    procreq
findit      <13582467
cp statable table.bak
rm    13582467
creatree    <1367.2458> 13672458
cp    136724.58    procreq
findit      <13672458
cp statable table.bak
rm    13672458
creatree    <1368.2457> 13682457
cp    136824.57    procreq
findit      <13682457
cp statable table.bak
rm    13682457
creatree    <1378.2456> 13782456
cp    137824.56    procreq
findit      <13782456
cp statable table.bak
rm    13782456
creatree    <1456.2378> 14562378
cp    145623.78    procreq
findit      <14562378
cp statable table.bak
rm    14562378
creatree    <1457.2368> 14572368
cp    145723.68    procreq
findit      <14572368
cp statable table.bak
rm    14572368
creatree    <1458.2367> 14582367
cp    145823.67    procreq
findit      <14582367
cp statable table.bak
rm    14582367
creatree    <1467.2358> 14672358
cp    146723.58    procreq
findit      <14672358
cp statable table.bak
rm    14672358
creatree    <1468.2357> 14682357
cp    146823.57    procreq
findit      <14682357
cp statable table.bak
rm    14682357
creatree    <1478.2356> 14782356
cp    147823.56    procreq
findit      <14782356
cp statable table.bak
rm    14782356
creatree    <1567.2348> 15672348
cp    156723.48    procreq
findit      <15672348
cp statable table.bak
rm    15672348
creatree    <1568.2347> 15682347
cp    156823.47    procreq
findit      <15682347
cp statable table.bak
rm    15682347
creatree    <1578.2346> 15782346
cp    157823.46    procreq
findit      <15782346
cp statable table.bak
rm    15782346
creatree    <1678.2345> 16782345
cp    167823.45    procreq
findit      <16782345
cp statable table.bak
rm    16782345
creatree    <2345.1678> 23451678
cp    234516.78    procreq
findit      <23451678
cp statable table.bak
rm    23451678
creatree    <2346.1578> 23461578
cp    234615.78    procreq
findit      <23461578
cp statable table.bak
rm    23461578
creatree    <2347.1568> 23471568
cp    234715.68    procreq
findit      <23471568
cp statable table.bak
```

Mar 16 17:05 1982 runall Page 3

```
rm    23471568
creatree    <2348.1567> 23481567
cp    234815.67    procreq
findit      <23481567
cp statable table.bak
rm    23481567
creatree    <2356.1478> 23561478
cp    235614.78    procreq
findit      <23561478
cp statable table.bak
rm    23561478
creatree    <2357.1468> 23571468
cp    235714.68    procreq
findit      <23571468
cp statable table.bak
rm    23571468
creatree    <2358.1467> 23581467
cp    235814.67    procreq
findit      <23581467
cp statable table.bak
rm    23581467
creatree    <2367.1458> 23671458
cp    236714.58    procreq
findit      <23671458
cp statable table.bak
rm    23671458
creatree    <2368.1457> 23681457
cp    236814.57    procreq
findit      <23681457
cp statable table.bak
rm    23681457
creatree    <2378.1456> 23781456
cp    237814.56    procreq
findit      <23781456
cp statable table.bak
rm    23781456
creatree    <2456.1378> 24561378
cp    245613.78    procreq
findit      <24561378
cp statable table.bak
rm    24561378
creatree    <2457.1368> 24571368
cp    245713.68    procreq
findit      <24571368
cp statable table.bak
rm    24571368
creatree    <2458.1367> 24581367
cp    245813.67    procreq
findit      <24581367
cp statable table.bak
rm    24581367
creatree    <2467.1358> 24671358
cp    246713.58    procreq
findit      <24671358
cp statable table.bak
rm    24671358
creatree    <2468.1357> 24681357
cp    246813.57    procreq
findit      <24681357
cp statable table.bak
rm    24681357
creatree    <2478.1356> 24781356
cp    247813.56    procreq
findit      <24781356
cp statable table.bak
rm    24781356
creatree    <2567.1348> 25671348
cp    256713.48    procreq
findit      <25671348
cp statable table.bak
rm    25671348
creatree    <2568.1347> 25681347
cp    256813.47    procreq
findit      <25681347
cp statable table.bak
rm    25681347
creatree    <2578.1346> 25781346
cp    257813.46    procreq
findit      <25781346
cp statable table.bak
rm    25781346
creatree    <2678.1345> 26781345
cp    267813.45    procreq
findit      <26781345
cp statable table.bak
rm    26781345
creatree    <3456.1278> 34561278
cp    345612.78    procreq
findit      <34561278
cp statable table.bak
rm    34561278
creatree    <3457.1268> 34571268
cp    345712.68    procreq
findit      <34571268
cp statable table.bak
rm    34571268
creatree    <3458.1267> 34581267
cp    345812.67    procreq
findit      <34581267
cp statable table.bak
rm    34581267
creatree    <3467.1258> 34671258
cp    346712.58    procreq
findit      <34671258
cp statable table.bak
rm    34671258
creatree    <3468.1257> 34681257
cp    346812.57    procreq
findit      <34681257
cp statable table.bak
rm    34681257
creatree    <3478.1256> 34781256
```

```
Mar 16 17:05 1982  runall Page 4

cp    347812.56    procreq
findit      <34781256
cp statable table.bak
rm    34781256
creatree    <3567.1248> 35671248
cp    356712.48    procreq
findit      <35671248
cp statable table.bak
rm    35671248
creatree    <3568.1247> 35681247
cp    356812.47    procreq
findit      <35681247
cp statable table.bak
rm    35681247
creatree    <3578.1246> 35781246
cp    357812.46    procreq
findit      <35781246
cp statable table.bak
rm    35781246
creatree    <3678.1245> 36781245
cp    367812.45    procreq
findit      <36781245
cp statable table.bak
rm    36781245
creatree    <4567.1238> 45671238
cp    456712.38    procreq
findit      <45671238
cp statable table.bak
rm    45671238
creatree    <4568.1237> 45681237
cp    456812.37    procreq
findit      <45681237
cp statable table.bak
rm    45681237
creatree    <4578.1236> 45781236
cp    457812.36    procreq
findit      <45781236
cp statable table.bak
rm    45781236
creatree    <4678.1235> 46781235
cp    467812.35    procreq
findit      <46781235
cp statable table.bak
rm    46781235
creatree    <5678.1234> 56781234
cp    567812.34    procreq
findit      <56781234
cp statable table.bak
rm    56781234
creatree    <8645.3127> 86453127
cp    864531.27    procreq
findit      <86453127
cp statable table.bak
rm    86453127
creatree    <7435.6128> 74356128
cp    743561.28    procreq
findit      <74356128
cp statable table.bak
rm    74356128
creatree    <7385.4126> 73854126
cp    738541.26    procreq
findit      <73854126
cp statable table.bak
rm    73854126
creatree    <8764.3125> 87643125
cp    876431.25    procreq
findit      <87643125
cp statable table.bak
rm    87643125
creatree    <7653.2418> 76532418
cp    765324.18    procreq
findit      <76532418
cp statable table.bak
rm    76532418
creatree    <1784.5623> 17845623
cp    178456.23    procreq
findit      <17845623
cp statable table.bak
rm    17845623
creatree    <3687.5124> 36875124
cp    368751.24    procreq
findit      <36875124
cp statable table.bak
rm    36875124
creatree    <4873.2156> 48732156
cp    487321.56    procreq
findit      <48732156
cp statable table.bak
rm    48732156
creatree    <2784.5136> 27845136
cp    278451.36    procreq
findit      <27845136
cp statable table.bak
rm    27845136
creatree    <2485.6137> 24856137
cp    248561.37    procreq
findit      <24856137
cp statable table.bak
rm    24856137
creatree    <3482.6157> 34826157
cp    348261.57    procreq
findit      <34826157
cp statable table.bak
rm    34826157
creatree    <2386.7145> 23867145
cp    238671.45    procreq
findit      <23867145
cp statable table.bak
rm    23867145
creatree    <2684.7135> 26847135
cp    268471.35    procreq
findit      <26847135
```

```
Mar 16 17:05 1982  runall Page 5

cp statable table.bak
rm    26847135
creatree    <2357.8146> 23578146
cp    235781.46    procreq
findit      <23578146
cp statable table.bak
rm    23578146
creatree    <2456.7138> 24567138
cp    245671.38    procreq
findit      <24567138
cp statable table.bak
rm    24567138
creatree    <2345.6178> 23456178
cp    234561.78    procreq
findit      <23456178
cp statable table.bak
rm    23456178
creatree    <1283.4567> 12834567
cp    128345.67    procreq
findit      <12834567
cp statable table.bak
rm    12834567
creatree    <1382.4567> 13824567
cp    138245.67    procreq
findit      <13824567
cp statable table.bak
rm    13824567
creatree    <1482.3567> 14823567
cp    148235.67    procreq
findit      <14823567
cp statable table.bak
rm    14823567
creatree    <1582.3467> 15823467
cp    158234.67    procreq
findit      <15823467
cp statable table.bak
rm    15823467
creatree    <1823.4567> 18234567
cp    182345.67    procreq
findit      <18234567
cp statable table.bak
rm    18234567
```

Input files to TREE.P to create trees to be.search by SEARCH.F

1234.5678
123456.78
1235.4678
123546.78
1236.4578
123645.78
1237.4568
123745.68
1238.4567
123845.67
1245.3678
124536.78
1246.3578
124635.78
1247.3568
124735.68
1248.3567
124835.67
1256.3478
125634.78
1257.3468
125734.68
1258.3467
125834.67
1267.3458
126734.58
1268.3457
126834.57
1278.3456
127834.56
1283.4567
128345.67
1345.2678
134526.78
1346.2578
134625.78
1347.2568
134725.68
1348.2567
134825.67
1356.2478
135624.78
1357.2468
135724.68
1358.2467
135824.67
1367.2458
136724.58
1368.2457
136824.57
1378.2456
137824.56
1382.4567
138245.67
1456.2378
145623.78
1457.2368
145723.68
1458.2367
145823.67
1467.2358
146723.58
1468.2357
146823.57
1478.2356
147823.56
1482.3567
148235.67
1567.2348
156723.48
1568.2347
156823.47
1578.2346
157823.46
1582.3467
158234.67
1678.2345
167823.45
1784.5623
178456.23
1823.4567
182345.67
2345.1678
2345.6178
234516.78
234561.78
2346.1578
234615.78
2347.1568
234715.68
2348.1567
234815.67
2356.1478
235614.78
2357.1468
2357.8146
235714.68
235781.46
2358.1467
235814.67
2367.1458
236714.58
2368.1457
236814.57
2378.1456
237814.56
2386.7145
238671.45

2456.1378
2456.7138
245613.78
245671.38
2457.1368
245713.68
2458.1367
245813.67
2467.1358
246713.58
2468.1357
246813.57
2478.1356
247813.56
2485.6137
248561.37
2567.1348
256713.48
2568.1347
256813.47
2578.1346
257813.46
2678.1345
267813.45
2684.7135
268471.35
2734.6815
273468.15
2784.5136
278451.36
2857.6314
285763.14
3456.1278
345612.78
3457.1268
345712.68
3458.1267
345812.67
3467.1258
346712.58
3468.1257
346812.57
3478.1256
347812.56
3482.6157
348261.57
3567.1248
356712.48
3568.1247
356812.47
3578.1246
357812.46
3678.1245
367812.45
3687.5124
368751.24
4567.1238
456712.38
4568.1237
456812.37
4578.1236
457812.36
4678.1235
467812.35
4873.2156
487321.56
5678.1234
567812.34
7385.4126
738541.26
7435.6128
743561.28
7543.2816
754328.16
7653.2418
765324.18
8356.2471
835624.71
8543.2617
854326.17
8567.2413
856724.13
8645.3127
864531.27
8756.4312
875643.12
8764.3125
876431.25

CENTRALIZED SCHEDULING ALGORITHM'S STATISTICS (CUBE & OMEGA N=8)

Average # of processors allocated

Processor	Resource: 1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.89796	1.97959	1.99592	2.00000	2.00000	2.00000	2.00000
3	1.00000	1.97959	2.89796	2.97959	3.00000	3.00000	3.00000	3.00000
4	1.00000	1.99592	2.97959	3.90041	3.97959	3.99592	4.00000	4.00000
5	1.00000	2.00000	3.00000	3.97959	4.89796	4.97959	5.00000	5.00000
6	1.00000	2.00000	3.00000	3.99592	4.97959	5.89796	6.00000	6.00000
7	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	7.00000
8	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	8.00000

Variance of processors allocated

Processor	Resource: 1	2	3	4	5	6	7	8
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.09163	.01999	.00406	.00000	.00000	.00000	.00000
3	.00000	.01999	.09163	.01999	.00000	.00000	.00000	.00000
4	.00000	.00406	.01999	.09784	.01999	.00407	.00000	.00000
5	.00000	.00000	.00000	.01999	.09163	.01999	.00000	.00000
6	.00000	.00000	.00000	.00407	.01999	.09163	.00000	.00000
7	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Standard Deviation of processors allocations

Processor	Resource: 1	2	3	4	5	6	7	8
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.30270	.14139	.06376	.00000	.00000	.00000	.00000
3	.00000	.14139	.30270	.14139	.00000	.00000	.00000	.00000
4	.00000	.06376	.14139	.31279	.14139	.06377	.00000	.00000
5	.00000	.00000	.00000	.14139	.30270	.14139	.00000	.00000
6	.00000	.00000	.00000	.06377	.14139	.30270	.00000	.00000
7	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Average percentage of processors allocated

Processor	Resource: 1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	.50000	.94898	.98980	.99796	1.00000	1.00000	1.00000	1.00000
3	.33333	.65986	.96599	.99320	1.00000	1.00000	1.00000	1.00000
4	.25000	.49898	.74490	.97510	.99490	.99898	1.00000	1.00000
5	.20000	.40000	.60000	.79592	.97959	.99592	1.00000	1.00000
6	.16667	.33333	.50000	.66599	.82993	.98299	1.00000	1.00000
7	.14286	.28571	.42857	.57143	.71429	.85714	1.00000	1.00000
8	.12500	.25000	.37500	.50000	.62500	.75000	.87500	1.00000

Average processor blocking probabilities

Processor	Resource: 1	2	3	4	5	6	7	8	
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000	2040.
2	.50000	.05102	.01020	.00204	.00000	.00000	.00000	.00000	7140.
3	.66667	.34014	.03401	.00680	.00000	.00000	.00000	.00000	14280.
4	.75000	.50102	.25510	.02490	.00510	.00102	.00000	.00000	17850.
5	.80000	.60000	.40000	.20408	.02041	.00408	.00000	.00000	14280.
6	.83333	.66667	.50000	.33401	.17007	.01701	.00000	.00000	7140.
7	.85714	.71429	.57143	.42857	.28571	.14286	.00000	.00000	2040.
8	.87500	.75000	.62500	.50000	.37500	.25000	.12500	.00000	255.

Total = 65025 cases

CENTRALIZED SCHEDULING HEURISTICS' STATISTICS WITH NO RETRY (CUBE N=8)

Resource: / Processor	1	2	3	4	5	6	7	8
Average # of processors allocated								
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.89796	1.91327	1.93469	1.95408	1.96939	1.98214	2.00000
3	1.00000	1.85714	2.73469	2.78367	2.83929	2.89541	2.94643	3.00000
4	1.00000	1.80000	2.67245	3.53714	3.63878	3.76224	3.88571	4.00000
5	1.00000	1.74490	2.56122	3.46633	4.34184	4.55102	4.78571	5.00000
6	1.00000	1.69388	2.44388	3.31224	4.27423	5.23980	5.62500	6.00000
7	1.00000	1.64286	2.33929	3.14286	4.08036	5.18750	6.37500	7.00000
8	1.00000	1.57143	2.32143	2.88571	3.92857	4.89286	6.37500	8.00000
Variance of processors allocated								
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.09163	.07921	.06104	.04381	.02968	.01754	.00000
3	.00000	.12245	.20512	.18177	.14509	.09875	.05070	.00000
4	.00000	.16000	.24067	.32372	.28788	.21592	.10837	.00000
5	.00000	.19002	.28707	.36111	.43417	.36984	.20408	.00000
6	.00000	.21241	.31828	.40658	.47709	.51903	.34152	.00000
7	.00000	.22959	.33131	.45102	.51140	.56306	.48438	.00000
8	.00000	.24490	.36097	.44408	.49490	.66709	.48438	.00000
Standard Deviation of processors allocations								
1	.00000	.00000	.00000	* .00000	.00000	.00000	.00000	.00000
2	.00000	.30270	.28145	.24707	.20931	.17226	.13243	.00000
3	.00000	.34993	.45291	.42635	.38091	.31425	.22517	.00000
4	.00000	.40000	.49058	.56897	.53655	.46467	.32919	.00000
5	.00000	.43592	.53579	.60093	.65891	.60815	.45176	.00000
6	.00000	.46088	.56416	.63764	.69072	.72044	.58440	.00000
7	.00000	.47916	.57560	.67158	.71512	.75037	.69597	.00000
8	.00000	.49487	.60081	.66639	.70349	.81676	.69597	.00000
Average percentage of processors allocated								
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	.50000	.94898	.95663	.96735	.97704	.98469	.99107	1.00000
3	.33333	.61905	.91156	.92789	.94643	.96514	.98214	1.00000
4	.25000	.45000	.66811	.88429	.90969	.94056	.97143	1.00000
5	.20000	.34898	.51224	.69327	.86837	.91020	.95714	1.00000
6	.16667	.28231	.40731	.55204	.71237	.87330	.93750	1.00000
7	.14286	.23469	.33418	.44898	.58291	.74107	.91071	1.00000
8	.12500	.19643	.29018	.36071	.49107	.61161	.79688	1.00000
Average processor blocking probabilities								
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000 2040 cases
2	.50000	.05102	.04337	.03265	.02296	.01531	.00893	.00000 7140 cases
3	.66667	.38095	.08844	.07211	.05357	.03486	.01786	.00000 14280 cases
4	.75000	.55000	.33189	.11571	.09031	.05944	.02857	.00000 17850 cases
5	.80000	.65102	.48776	.30673	.13163	.08980	.04286	.00000 14280 cases
6	.83333	.71769	.59269	.44796	.28763	.12670	.06250	.00000 7140 cases
7	.85714	.76531	.66582	.55102	.41709	.25893	.08929	.00000 2040 cases
8	.87500	.80357	.70982	.63929	.50893	.38839	.20313	.00000 255 cases
								Total = 65025 cases

CENTRALIZED SCHEDULING HEURISTICS' STATISTICS WITH NO RETRY (OMEGA N=0)

Average # of processors allocated

Resource: / Processor	1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.89796	1.85714	1.80000	1.74490	1.69388	1.64286	1.57143
3	1.00000	1.91327	2.73469	2.67245	2.56122	2.44388	2.33929	2.32143
4	1.00000	1.93469	2.78367	3.53714	3.46633	3.31224	3.14286	2.88571
5	1.00000	1.95408	2.83929	3.63878	4.34056	4.27423	4.08036	3.92857
6	1.00000	1.96939	2.89541	3.76224	4.55102	5.23980	5.18750	4.89286
7	1.00000	1.98214	2.94643	3.88571	4.78571	5.62500	6.37500	6.37500
8	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	8.00000

Variance of processors allocated

1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.09163	.12245	.16000	.19002	.21241	.22959	.24490
3	.00000	.07921	.20512	.24067	.28707	.31828	.33131	.36097
4	.00000	.06104	.18177	.32372	.36111	.40658	.45102	.44408
5	.00000	.04381	.14509	.28788	.43631	.47709	.51140	.49490
6	.00000	.02968	.09875	.21592	.36984	.51903	.56306	.66709
7	.00000	.01754	.05070	.10837	.20408	.34152	.48438	.48438
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Standard Deviation of processors allocations

1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.30270	.34993	.40000	.43592	.46088	.47916	.49487
3	.00000	.28145	.45291	.49058	.53579	.56416	.57560	.60081
4	.00000	.24707	.42635	.56897	.60093	.63764	.67158	.66639
5	.00000	.20931	.38091	.53655	.66054	.69072	.71512	.70349
6	.00000	.17226	.31425	.46467	.60818	.72044	.75037	.81676
7	.00000	.13243	.22517	.32919	.45176	.58440	.69597	.69597
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Average percentage of processors allocated

1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	.50000	.94898	.92857	.90000	.87245	.84694	.82143	.78571
3	.33333	.63776	.91156	.89082	.85374	.81463	.77976	.77381
4	.25000	.48367	.69592	.88429	.86658	.82806	.78571	.72143
5	.20000	.39082	.56786	.72776	.86811	.85485	.81607	.78571
6	.16667	.32823	.48257	.62704	.75850	.87330	.86458	.81548
7	.14286	.28316	.42092	.55510	.68367	.80357	.91071	.91071
8	.12500	.25000	.37500	.50000	.62500	.75000	.87500	1.00000

Average processor blocking probabilities

1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000	2040 cases
2	.50000	.05102	.07143	.10000	.12755	.15306	.17857	.21429	7140 cases
3	.66667	.36224	.08844	.10918	.14626	.18537	.22024	.22619	14280 cases
4	.75000	.51633	.30408	.11571	.13342	.17194	.21429	.27857	17850 cases
5	.80000	.60918	.43214	.27224	.13189	.14515	.18393	.21429	14280 cases
6	.83333	.67177	.51743	.37296	.24150	.12670	.13542	.18452	7140 cases
7	.85714	.71684	.57908	.44490	.31633	.19643	.08929	.08929	2040 cases
8	.87500	.75000	.62500	.50000	.37500	.25000	.12500	.00000	255 cases
								Total =	65025 cases

CENTRALIZED SCHEDULING HEURISTICS' STATISTICS WITH RETRY (CUBE N=8)

Average # of processors allocated

Processor \ Resource	1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.89796	1.97959	1.99592	2.00000	2.00000	2.00000	2.00000
3	1.00000	1.97959	2.81888	2.95204	2.98852	2.99745	3.00000	3.00000
4	1.00000	1.99592	2.95102	3.66204	3.90439	3.97857	4.00000	4.00000
5	1.00000	2.00000	2.98724	3.91531	4.60842	4.88010	4.96429	5.00000
6	1.00000	2.00000	2.99745	3.97551	4.88393	5.54337	5.91071	6.00000
7	1.00000	2.00000	3.00000	3.99643	4.97768	5.91071	6.68750	7.00000
8	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	8.00000

Variance of processors allocated

Processor \ Resource	1	2	3	4	5	6	7	8
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.09163	.01999	.00406	.00000	.00000	.00000	.00000
3	.00000	.01999	.14832	.04566	.01135	.00254	.00000	.00000
4	.00000	.00406	.04658	.23191	.08631	.02097	.00000	.00000
5	.00000	.00000	.01259	.07752	.25100	.10552	.03444	.00000
6	.00000	.00000	.00254	.02389	.10260	.26343	.08131	.00000
7	.00000	.00000	.00000	.00356	.02182	.08131	.21484	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Standard Deviation of processors allocations

Processor \ Resource	1	2	3	4	5	6	7	8
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.30270	.14139	.06376	.00000	.00000	.00000	.00000
3	.00000	.14139	.38512	.21368	.10653	.05044	.00000	.00000
4	.00000	.06376	.21582	.48157	.29378	.14481	.00000	.00000
5	.00000	.00000	.11222	.27843	.50100	.32484	.18557	.00000
6	.00000	.00000	.05044	.15457	.32031	.51325	.28516	.00000
7	.00000	.00000	.00000	.05967	.14772	.28516	.46351	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Average percentage of processors allocated

Processor \ Resource	1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	.50000	.94898	.98980	.99796	1.00000	1.00000	1.00000	1.00000
3	.33333	.65986	.93963	.98401	.99617	.99915	1.00000	1.00000
4	.25000	.49898	.73776	.91551	.97615	.99464	1.00000	1.00000
5	.20000	.40000	.59745	.78306	.92168	.97602	.99286	1.00000
6	.16667	.33333	.49957	.66259	.81399	.92389	.98512	1.00000
7	.14286	.28571	.42857	.57092	.71110	.84439	.95536	1.00000
8	.12500	.25000	.37500	.50000	.62500	.75000	.87500	1.00000

Average processor blocking probabilities

Processor \ Resource	1	2	3	4	5	6	7	8	
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000	2040 cases
2	.50000	.05102	.01020	.00204	.00000	.00000	.00000	.00000	7140 cases
3	.66667	.34014	.06037	.01599	.00383	.00085	.00000	.00000	14280 cases
4	.75000	.50102	.26224	.08449	.02385	.00536	.00000	.00000	17850 cases
5	.80000	.60000	.40255	.21694	.07832	.02398	.00714	.00000	14280 cases
6	.83333	.66667	.50043	.33741	.18601	.07611	.01488	.00000	7140 cases
7	.85714	.71429	.57143	.42908	.28890	.15561	.04464	.00000	2040 cases
8	.87500	.75000	.62500	.50000	.37500	.25000	.12500	.00000	255 cases
								Total =	65025 cases

CENTRALIZED SCHEDULING HEURISTICS' STATISTICS WITH RETRY (OMEGA N=8)

Resource: / Processor	1	2	3	4	5	6	7	8
Average # of processors allocated								
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.89796	1.97959	1.99592	2.00000	2.00000	2.00000	2.00000
3	1.00000	1.91327	2.81888	2.95102	2.98724	2.99745	3.00000	3.00000
4	1.00000	1.93469	2.85204	3.66204	3.91837	3.97551	3.99643	4.00000
5	1.00000	1.95408	2.89413	3.73163	4.60842	4.89668	4.97768	5.00000
6	1.00000	1.96939	2.93622	3.81531	4.75255	5.55102	5.91071	6.00000
7	1.00000	1.98214	2.97321	3.90357	4.88393	5.77679	6.71875	7.00000
8	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	8.00000
Variance of processors allocated								
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.09163	.01999	.00406	.00000	.00000	.00000	.00000
3	.00000	.07921	.14832	.04658	.01259	.00254	.00000	.00000
4	.00000	.06104	.12607	.23191	.07497	.02389	.00356	.00000
5	.00000	.04381	.09466	.20247	.25100	.09264	.02182	.00000
6	.00000	.02968	.05971	.15262	.19132	.26270	.08131	.00000
7	.00000	.01754	.02607	.08713	.10260	.17339	.20215	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
Standard Deviation of processors allocations								
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.30270	.14139	.06376	.00000	.00000	.00000	.00000
3	.00000	.28145	.38512	.21582	.11222	.05044	.00000	.00000
4	.00000	.24707	.35506	.48157	.27380	.15457	.05967	.00000
5	.00000	.20931	.30767	.44997	.50100	.30437	.14772	.00000
6	.00000	.17226	.24435	.39067	.43740	.51254	.28516	.00000
7	.00000	.13243	.16146	.29518	.32031	.41640	.44961	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
Average percentage of processors allocated								
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	.50000	.94898	.98980	.99796	1.00000	1.00000	1.00000	1.00000
3	.33333	.63776	.93963	.98367	.99575	.99915	1.00000	1.00000
4	.25000	.48367	.71301	.91551	.97959	.99388	.99911	1.00000
5	.20000	.39082	.57883	.74633	.92168	.97934	.99554	1.00000
6	.16667	.32823	.48937	.63588	.79209	.92517	.98512	1.00000
7	.14286	.28316	.42474	.55765	.69770	.82526	.95982	1.00000
8	.12500	.25000	.37500	.50000	.62500	.75000	.87500	1.00000
Average processor blocking probabilities								
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.50000	.05102	.01020	.00204	.00000	.00000	.00000	.00000
3	.66667	.36224	.06037	.01633	.00425	.00085	.00000	.00000
4	.75000	.51633	.28699	.08449	.02041	.00612	.00089	.00000
5	.80000	.60918	.42117	.25367	.07832	.02066	.00446	.00000
6	.83333	.67177	.51063	.36412	.20791	.07483	.01488	.00000
7	.85714	.71684	.57526	.44235	.30230	.17474	.04018	.00000
8	.87500	.75000	.62500	.50000	.37500	.25000	.12500	.00000

2040	cases
7140	cases
14280	cases
17850	cases
14280	cases
7140	cases
2040	cases
255	cases
Total = 65025	cases

DISTRIBUTED SCHEDULING ALGORITHM'S STATISTICS (CUBE & OMEGA N=8)

Average # of processors allocated

Processor	Resource: 1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	1.00000	1.89796	1.97959	1.99592	2.00000	2.00000	2.00000	2.00000
3	1.00000	1.97959	2.57143	2.83265	2.93878	2.98980	3.00000	3.00000
4	1.00000	1.99592	2.76735	3.31265	3.63673	3.81633	3.91429	4.00000
5	1.00000	2.00000	2.86735	3.52245	4.04082	4.44898	4.75000	5.00000
6	1.00000	2.00000	2.94898	3.71837	4.38775	4.97959	5.50000	6.00000
7	1.00000	2.00000	3.00000	3.88571	4.71429	5.50000	6.25000	7.00000
8	1.00000	2.00000	3.00000	4.00000	5.00000	6.00000	7.00000	8.00000

Variance of processors allocated

Processor	Resource: 1	2	3	4	5	6	7	8
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.09163	.01999	.00406	.00000	.00000	.00000	.00000
3	.00000	.01999	.24490	.13934	.05748	.01010	.00000	.00000
4	.00000	.00406	.17853	.32755	.26396	.14994	.07837	.00000
5	.00000	.00000	.11506	.24950	.36568	.32903	.18750	.00000
6	.00000	.00000	.04842	.20232	.35985	.38734	.25000	.00000
7	.00000	.00000	.00000	.10122	.20408	.25000	.18750	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Standard Deviation of processors allocations

Processor	Resource: 1	2	3	4	5	6	7	8
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.00000	.30270	.14139	.06376	.00000	.00000	.00000	.00000
3	.00000	.14139	.49487	.37329	.23974	.10050	.00000	.00000
4	.00000	.06376	.42252	.57232	.51377	.38722	.27994	.00000
5	.00000	.00000	.33920	.49950	.60472	.57361	.43301	.00000
6	.00000	.00000	.22004	.44979	.59988	.62237	.50000	.00000
7	.00000	.00000	.00000	.31816	.45175	.50000	.43301	.00000
8	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000

Average percentage of processors allocated

Processor	Resource: 1	2	3	4	5	6	7	8
1	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
2	.50000	.94898	.98980	.99796	1.00000	1.00000	1.00000	1.00000
3	.33333	.65986	.85714	.94422	.97959	.99660	1.00000	1.00000
4	.25000	.49898	.69184	.82816	.90918	.95408	.97857	1.00000
5	.20000	.40000	.57347	.70449	.80816	.88980	.95000	1.00000
6	.16667	.33333	.49150	.61973	.73129	.82993	.91667	1.00000
7	.14286	.28571	.42857	.55510	.67347	.78571	.89286	1.00000
8	.12500	.25000	.37500	.50000	.62500	.75000	.87500	1.00000

Average processor blocking probabilities

Processor	Resource: 1	2	3	4	5	6	7	8	
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000	2040 cases
2	.50000	.05102	.01020	.00204	.00000	.00000	.00000	.00000	7140 cases
3	.66667	.34014	.14286	.05578	.02041	.00340	.00000	.00000	14280 cases
4	.75000	.50102	.30816	.17184	.09082	.04592	.02143	.00000	17850 cases
5	.80000	.60000	.42653	.29551	.19184	.11020	.05000	.00000	14280 cases
6	.83333	.66667	.50850	.38027	.26871	.17007	.08333	.00000	7140 cases
7	.85714	.71429	.57143	.44490	.32653	.21429	.10714	.00000	2040 cases
8	.87500	.75000	.62500	.50000	.37500	.25000	.12500	.00000	255 cases
								Total =	65025 cases

Resource:	1	2	3	4	5	6	7	8
Processor	**********	**********	**********	**********	**********	**********	**********	**********
	Average units of delaying time							
1	3.00000	3.00000	3.00000	3.00000	3.00000	3.00000	3.00000	3.00000
2	3.42857	3.91837	3.83673	3.63265	3.42857	3.26531	3.14286	3.00000
3	3.28571	4.10204	4.15646	3.99048	3.75850	3.52041	3.28571	3.00000
4	3.05714	3.98367	4.19388	4.11755	3.91429	3.65306	3.35714	3.00000
5	2.82857	3.75918	4.05918	4.06449	3.91429	3.67755	3.37857	3.00000
6	2.61905	3.50340	3.84864	3.91020	3.80952	3.61224	3.34524	3.00000
7	2.42857	3.24490	3.60204	3.70204	3.64796	3.50000	3.28571	3.00000
8	2.25000	3.00000	3.39286	3.54286	3.53571	3.42857	3.25000	3.00000
	Variance of delaying time							
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.53061	.85048	.66722	.50179	.32653	.19492	.12245	.00000
3	.23583	1.20727	.92110	.76408	.62875	.51149	.36281	.00000
4	.09673	.96708	.67976	.57720	.53959	.50004	.35102	.00000
5	.03918	.73303	.51160	.38588	.35592	.34092	.24240	.00000
6	.01361	.54931	.37902	.24953	.20635	.19658	.13081	.00000
7	.00000	.42982	.30519	.18352	.12097	.09111	.04082	.00000
8	.00000	.35714	.26531	.13388	.06122	.03061	.00000	.00000
	Standard Deviation of delaying time							
1	.00000	.00000	.00000	.00000	.00000	.00000	.00000	.00000
2	.72843	.92221	.81684	.70837	.57143	.44150	.34993	.00000
3	.48562	1.09876	.95974	.87412	.79294	.71518	.60234	.00000
4	.31102	.98340	.82447	.75974	.73457	.70714	.59247	.00000
5	.19795	.85617	.71526	.62119	.59659	.58389	.49234	.00000
6	.11664	.74115	.61564	.49953	.45426	.44338	.36168	.00000
7	.00000	.65561	.55244	.42840	.34780	.30184	.20203	.00000
8	.00000	.59761	.51508	.36589	.24744	.17496	.00000	.00000