A multiprocessor system using a switch matrix configuration

Rabah Aoufi

A MULTIPROCESSOR SYSTEM USING A SWITCH MATRIX CONFIGURATION

BY

RABAH AOUFI, 1955 -

A THESIS

Presented to the Faculty of the Graduate School of the

UNIVERSITY OF MISSOURI-ROLLA

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

1980

Approved by

[illegible] (Advisor) [illegible]

[illegible]

ABSTRACT

This thesis describes a class of interconnection networks based on the use of a switch matrix to provide processor to memory communication. This switch allows a direct link between any processor to any memory module. The cost and performance of this network are analytically examined. The results are compared with those of a multiprocessor system using a time-shared bus configuration and it is shown that for the two extreme cases of maximum and minimum throughput, the two approaches are equivalent from a performance point of view. However, in the general case, even with a higher cost, the switch matrix provides a much better performance than the time-shared bus configuration. Furthermore, the architecture of a multiprocessor MIMD type computer using a switch matrix is investigated and Petri net techniques are used to model process coordination among processors.

ACKNOWLEDGMENT

I would like to express my gratitude to Dr. Darrow Dawson for his guidance during my graduate work at the University of Missouri-Rolla. I also would like to thank Dr. Theodore E. McCracken and Dr. Min Ming Tang for serving on my Master's Committee and Mrs. Monique Helterbrand for her precision and promptness in preparing the typescript.

A special thanks to my mother, Sahara, for her patience, understanding and moral support during my stay in the United States of America.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

I. INTRODUCTION

A. REVIEW OF MULTIPROCESSING SYSTEMS

During the past several years, multiprocessing systems have been discussed in the literature and a number of different systems have been implemented or proposed. Low cost microprocessors are now being designed into multiprocessing systems. Parallel (SIMD) type processors, computer networking and multiprocessor systems are among the existing organizations. This thesis considers the multiprocessor MIMD feature. SIMD and MIMD are defined in Section IV.

It is important to recall the fundamental definition of a multiprocessor system as given by [1], before the advantages of multiprocessor systems are presented.

A multiprocessor computer is a system containing two or more processor units of approximatively comparable capabilities. Each unit has access to shared common memory as well as having common access to at least a portion of the I/O devices. In addition all processor units are controlled by one operating system that provides interaction between the processors and the programs they are executing at all levels.

Several advantages may be realized with multiprocessor systems. Throughput often increases almost directly with the number of processors while system cost increases by only a small amount. Shared resources provide economic advantage by eliminating devices to be duplicated in other systems. On the other hand they provide direct access of data without transmission from system to system. The cost of a standby unit is small and a spare processor can be switched into the system to replace a failed processor.

B. <u>CLASSIFICATION OF MULTIPROCESSOR SYSTEMS</u>

This section contains a brief discussion of the classification of multiprocessor systems. Two distinguishing features that differentiate between designs are the use of processing units and the interconnection of processor units and memories.

1. <u>Symmetric and Asymmetric Processor</u>: The symmetric multiprocessor system consists of a network of functionally equivalent processors. This type system is used in a general purpose environment, where processing requirements are constantly changing. The advantage of this class is that a given task can be assigned to any idle processor for execution, since there is an equivalence among individual processors. Another significant advantage of this class is that the failure of a module does not cause the failure of the entire machine. However, symmetric systems require that every processor have full capabilities (which increases hardware expense, and complicates the operating system which becomes responsible for the identification and control of tasks).

A second group consists of heterogenous processors specially configured for a set number of tasks. Tasks and their actions must be completely known in advance. In this case, processors may be specialized to carry out one particular type of task. One processor may perform all I/O operations, another provides floating point arithmetic capability, a third provides file maintenance. The operating system is greatly simplified and becomes a task scheduler.

2. System Organization: In addition to classifying multiprocessor systems according to processor use, they may be grouped in relation to the interconnection of processors with system memories and peripheral devices. Three main types of organizations are possible.

a. Switch Matrix: This scheme provides direct paths from any processor to any memory or peripheral. This allows many processors to simultaneously utilize many different memory modules, reducing memory reference interference between processors. However, the switching matrix may be extremely expensive (the cost increases rapidly with the number of processors) eliminating much of the cost advantage of a multiprocessor system.

b. Time-shared Bus: This method is to multiplex all processors memories and peripheral devices over one data bus. This is a lower cost approach, but system throughput becomes limited by bus capacity.

c. Multiport Memory Systems: In this third method each processor has access through its own bus to all memory modules. Like the two previous organizations, the multiport system organization has the disadvantage of high cost multiple-connection hardware.

All of the above organizations and their variations are useful and worthy of consideration. This thesis is concerned with the symmetric processor and its associated switching matrix. Much of the discussion can be applied to the other classes as well.

C. OUTLINE

The next section describes in detail the switch matrix organization. Section III analyzes performance and cost of a typical multiprocessor system. Such a system is the MIMD type computer treated in section IV where process coordination is modeled using Petri net techniques as a tool for the purpose.

II. SWITCH ORGANIZATION

A. PRINCIPLE OF OPERATION

A block diagram of the switch organization is shown in Figure 1.

1. Description: The switch that interconnects processors and data memories to allow memory sharing, consists of a number of nodes connected via ports. Each node contains two input ports labeled A and B and two output ports labeled C and D. Each node can send a message on its output ports and receive one on its input ports. It is assumed that each memory can respond to a single request during one cycle so that there is no simultaneous double service. The message contains the address of the memory to be mapped into physical memory and a priority word.

When a switch node receives a message, it attempts to route it correctly through the appropriate path. This is accomplished by storing in each node a table which maps the recipient address into the port number as shown in Table I. Input A could be connected to either the output labeled D or the output labeled C depending on the value of some generated control. However, input B could be connected only to the output labeled C. This technique reduces the complexity of the table mapping and defines a unique path between the message entry and its destination. It is clear that the inputs of the root-node can be switched to anyone of the outputs.

2. Contention: The hierarchy in priority is set by

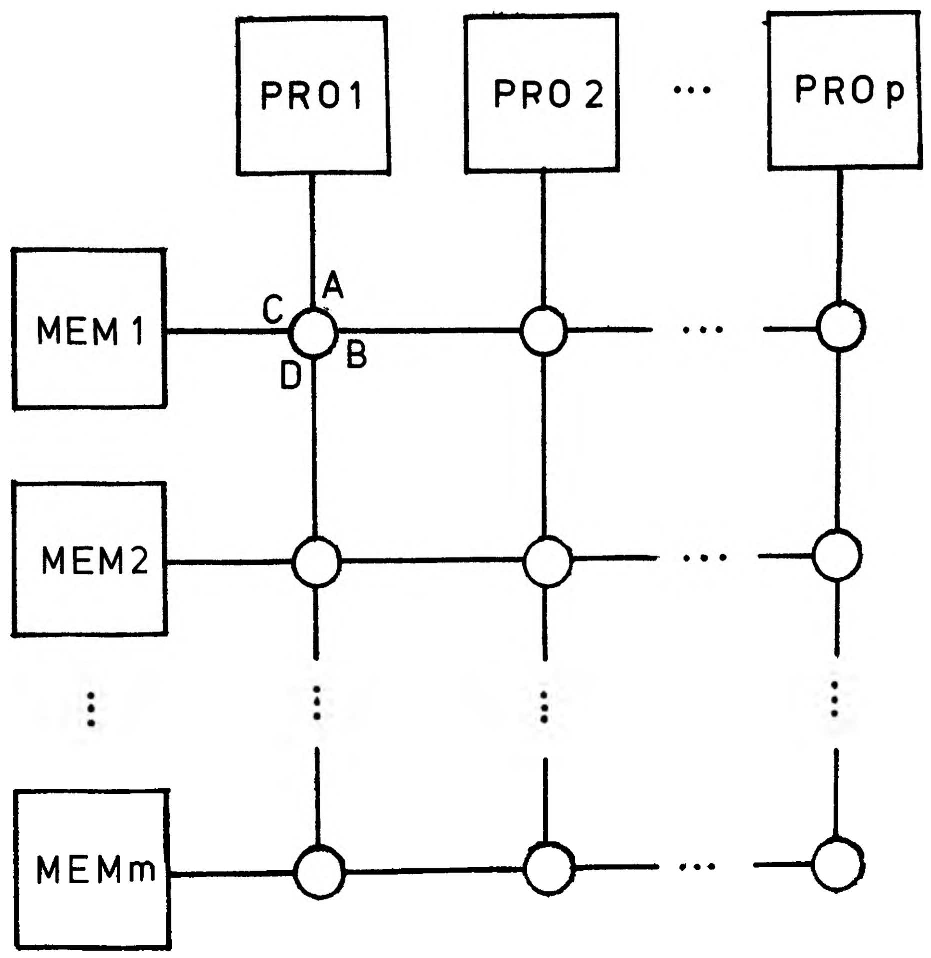

Figure 1. Multiprocessor Switch Organization

TABLE I

TABLE MAPPING MEMORY ADDRESSES INTO SWITCH NODES

PROCESSOR UNITS	SWITCH NODES	MEMORY MODULE
1	1,1	
2	2,1 - 1,1	
.	.	
.	.	1
.	.	
p	p,1 - 2,1 - ... - 1,1	
1	1,1 - 1,2	
2	2,1 - 2,2 - 1,2	
.	.	
.	.	
.	.	2
p	p,1 - p,2 - ... - 2,2 - 1,2	
.	.	.
.	.	.
.	.	.
1	1,1 - 1,2 - ... - 1,m	
2	2,1 - 2,2 - ... - 2,m - 1,m	m
.	.	
.	.	
.	.	
p	p,1 - p,2 - ... - p,m - ... - 2,m - 1,m	

incrementing the request priority as it passes through the node. Preference to route the request is given then to the request with highest priority. When a request is accepted, it is followed by latching sequentially the high order byte and then the low order byte of the memory address. A message is then at any instant distributed between two nodes and a conflict to acquire the node is created between the beginning of some request and the middle part of another. To avoid such a conflict, each part of the message should have the priority word associated with it. This requires an increase of the message word length. Figures 2 and 3 show a possible solution to the contention problem.

3. Reliability: When a request has been made by a processor to access a certain memory address, a signal message is reported back through the switch to the requesting processor to indicate whether the operation has been successful or not. In case there is a failure of the operation, another attempt will be made by the processor to achieve its request granting.

When a switch node fails, a misrouted message could be created. A leaking message should be inserted at the beginning of a request by every originator. This message leaks out the switch from the misrouted request. The leaking message consists of a message with a higher priority value enabling the processor to gain access to the node. The number of bits associated with each request word should be sufficient in order to prevent the priority value from reaching

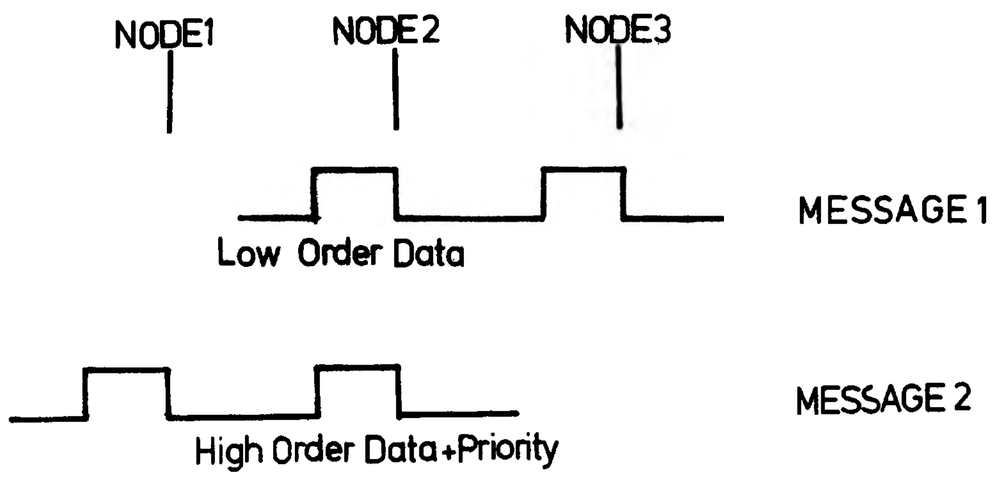

Figure 2. Conflict due to Absence of Priority Word

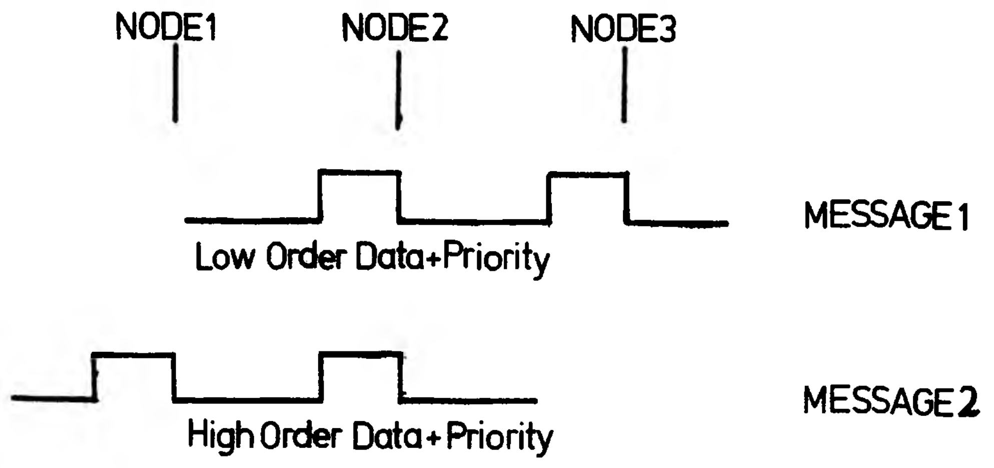

Figure 3. Solution to the Conflict Using Priority Word Presence

its maximum and overflowing.

B. CONTROL OF THE SWITCH

The functional block diagram of a switch node appears in Figure 4. All single lines in the figure are multiple bit lines. The double lines on INOUT box represent incoming and outgoing address and data lines. A read/write control line is also provided. The function of the INOUT box is to set up a connection between the incoming information port and one of the outgoing ones, according to the value of the input X. The input X may be encoded with as few as two bits as shown in Table II.

TABLE II

Input X Coding

Connection	Input X
A - C	01
A - D	10
B - C	11
B - D	Forbidden

The function of the CONTROL box is to generate the signal X and provide arbitration. A request is generated when its line is presented at the input port. The memory address is mapped into the stored table to provide the correct routing of the selected message. A signal X is then issued to box INOUT to specify the right exit port. When a request for a busy memory is rejected, a busy signal is eventually transmitted to the source which originated the blocked request. The DONE signal is supplied to each

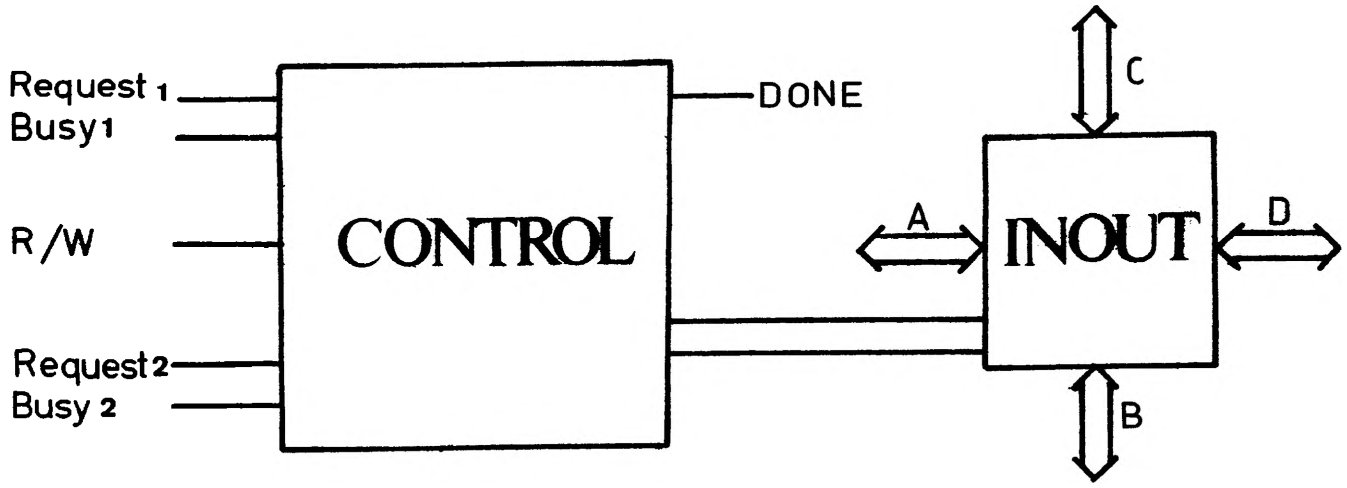

Figure 4. Block Diagram of a Switch Node

CONTROL box to guarantee information flow about the success or failure of operations. In case of failure, new attempts should be made till the operation is achieved correctly. To avoid any gate delay, the DONE signal is connected directly through the network.

Actual implementation of the switch in the real world requires additional practical considerations. An evaluation of this interconnection network in terms of system performance/cost and allowance of programming concurrency will be made in the next two sections.

III. PERFORMANCE AND COST OF THE SYSTEM

A. SYSTEM THROUGHPUT

The estimation of system performance and cost is motivated by the work of [2]. In his analysis, Reyling derived results based on the utilization of a time-sharing technique as shown in Figure 5. This section deals with the equivalent space-sharing technique. The analytical results, concerning system performance and cost, are compared with those of time-sharing technique and validated through examples.

Space-sharing means that a set of resources is partitioned into non-intersecting blocks such that each block executes some application. The applications are executed independently in parallel. Time-sharing reduces the idle time. Space-sharing reduces the percentage of resources that are idled.

To determine multiprocessor throughput as a function of the number of microprocessors required, the characteristics of the system have been defined as:

Ts: System throughput defined as the number of instructions executed per second by the system.

Tp: Throughput of an individual processor when there is no memory interference.

p: Number of processors in the system

m: Number of memory modules in the system.

The effects of interference when memory is used for making single-word transfers have been considered here;

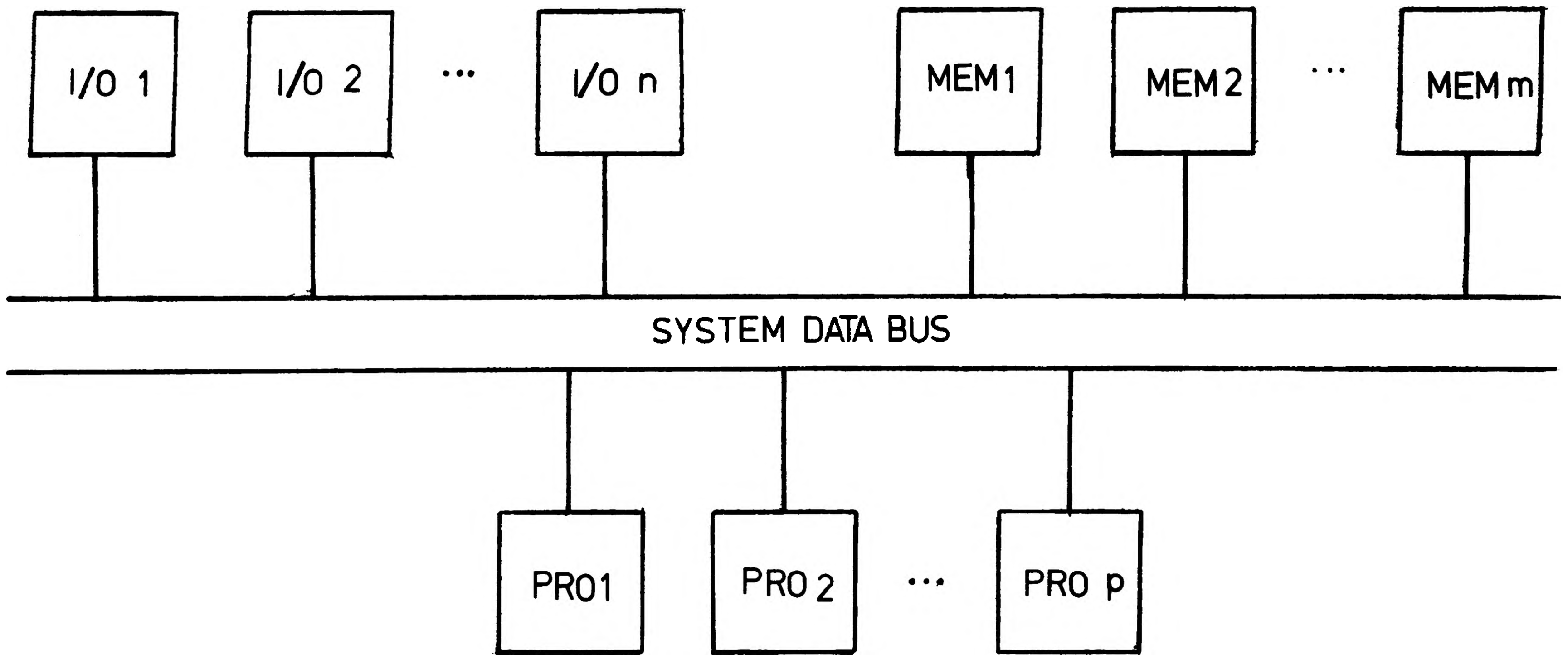

Figure 5. Multiprocessor Data Bus Organization

contention for multiple-word transfer units also affects throughput of a particular system and may be investigated in a manner similar to the following discussion.

When several processors simultaneously address the same memory module, a memory interference occurs. If n generated requests are queued to the same memory module, then n-1 processors must wait for the module to become unlocked in order to gain access to it. Throughput of the entire system is reduced because each processor is slowed down.

In order to study memory interference in more general terms, maximum, minimum, and average throughput Ts is determined. For this purpose, the following model is described. At a given instant of time t, p different requests are generated and divided among the m modules. It is assumed that the processing time is null. Furthermore it is assumed that a processor issues a new request immediately after receiving its current request with a uniform probability (1/m). To illustrate the ideas, an example with p=4 and m=4 is considered. The number of requests simultaneously present at memory module j at time t will be indicated by $X_t(j)$. In the case where:

$X_t(1) = 0$, $X_t(2) = 3$, $X_t(3) = 1$, $X_t(4) = 0$

the model will be illustrated by

p	0	1	2	3
m	2	2	2	3

1. Maximum Throughput: The maximum throughput Ts (MAX) will occur if each memory module receives a single request at a given instant of time t. In other terms,

$X_t(j) = 1 \qquad \text{for } j = 1, 2, \ldots, m$

In this case, all the processors are doing useful work since they are accessing different memory modules of the shared main memory. Clearly, Ts will equal pTp. This is shown graphically in Figures 6 and 7. This result is also true for time-sharing system performance.

2. Minimum Throughput: It is also of interest to find the minimum value of T_s. The worst possible case would be if all p requests had to be queued to the same memory module j, so that

$X_t(j) = P \qquad \text{for } j = 1, 2, \ldots, m$

and consequently, p-1 processors will be waiting to gain access to the resource. It is assumed that the probability that a request will be pending is also $(\frac{1}{m})$.

The memory bandwidth B is defined as the number of requests serviced per cycle. It follows that, for the above example, the bandwidth would be:

$$B = \frac{\text{number of processors}}{\text{number of cycles}} = \frac{4}{3} = 1.33$$

where the number of cycles is equal to $X_t(j)$ maximum for $j=1,\ldots,m$

The decrease in throughput could be derived by considering the ratio

$$R = \frac{\text{bandwidth with maximum interference}}{\text{bandwidth with no interference}}$$

expressing the fact that p-1 processors would be waiting for the busy memory module during interference yields

$$R = \frac{\frac{p}{1 + (p-1)(1/m)}}{\frac{p}{1}} = \frac{1}{1 + (p-1)(1/m)}$$

The minimum value of Ts is given as:

$$Ts\ (min) = (\text{throughput with no interference}) \times R$$
$$= p x T_p x\ R = \frac{p x T_p}{1 + (p-1)(1/m)}$$

This minimal value of throughput may be used to determine the range of possible throughputs and has been plotted in Figure 6 for m=3 and in Figure 7 for m=10. The two figures show that with the hypothesis stating that m=p, the results are equivalent to the time-sharing system performance results. Even with maximum interference, both analysis still depicts a substantial increase in Ts with p. However, it should be pointed out that these last two cases concerning performance bounds are events of small occurence. As an example, a system with parameter m=p=n has the random sampling probabilities

$$g(1) = \frac{n!}{n^n} \quad \text{and} \quad g(p) = \frac{1}{n^{n-1}}$$

where g(h) is the probability that $X_t(j)$ maximum is equal to h. For a 7x7 system, g(1) and g(7) are given by

$$g(1) = \frac{7!}{7^7} = 0.00611 \text{ and } g(7) = \frac{1}{7^6} = 0.00000849$$

As one can see, these probabilities are very low to let the maximum and minimum interference occur frequently.

3. <u>Average Throughput</u>: Average throughput is a

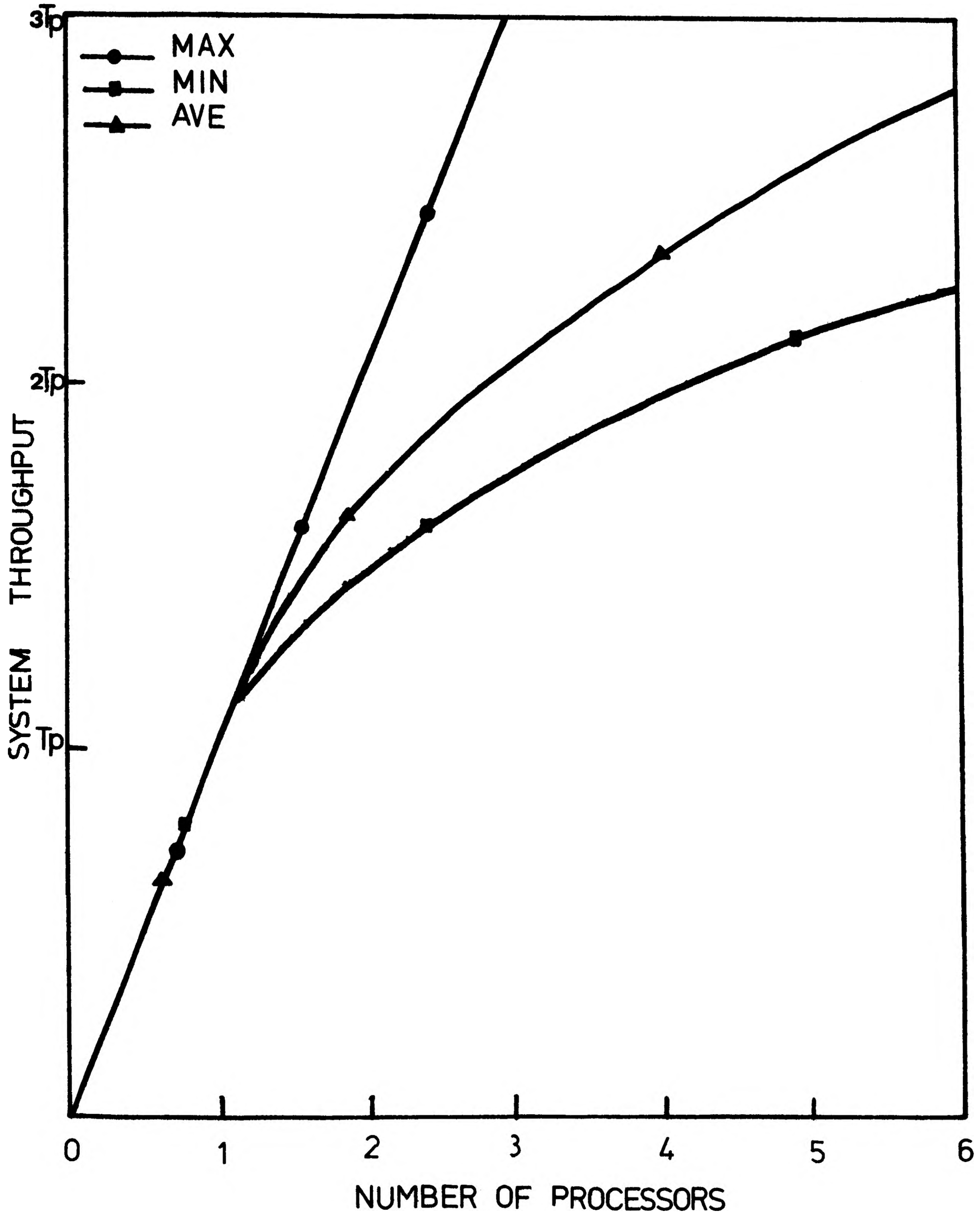

Figure 6. System Throughput Vs Number of Processors, m=3

deterministic factor of system performance. It is computed by considering a sequence of transition states viewed as a discrete Markovian process with state space (1,2,...,m) and with probability transition A(i,j) from state i to state j. Let p(i) denote the steady-state probability of state i. Then,

$$p(i) = \sum_{j=1}^{m} A(i,j)\, p(j), \qquad k=1,2,\ldots,m$$

To simplify the analysis, an assumption is made that all the states are inter-reachable. The number of busy modules is represented by a state of m-tuple (p1,p2,...,pm) with

$$\sum_{i=1}^{m} p_i = p$$

A new state $(j_1,j_2,\ldots,j_m)$ is reachable from state $(i_1,i_2,\ldots,i_m)$ with the transition probability [3]

$$\frac{x!}{(j_1-i_1)!\ldots(j_m-i_m)!} \cdot \left(\frac{1}{m}\right)^x$$

where x is the number of nonzero elements in the new state vector. Furthermore, the distribution probability p(i) of all possible states obeys the normalizing equation

$$\sum_{i=1}^{m} p(i) = 1$$

In order to compute the elements of the transition matrix A(i,j), the enumeration tree of a 4x4 system as in

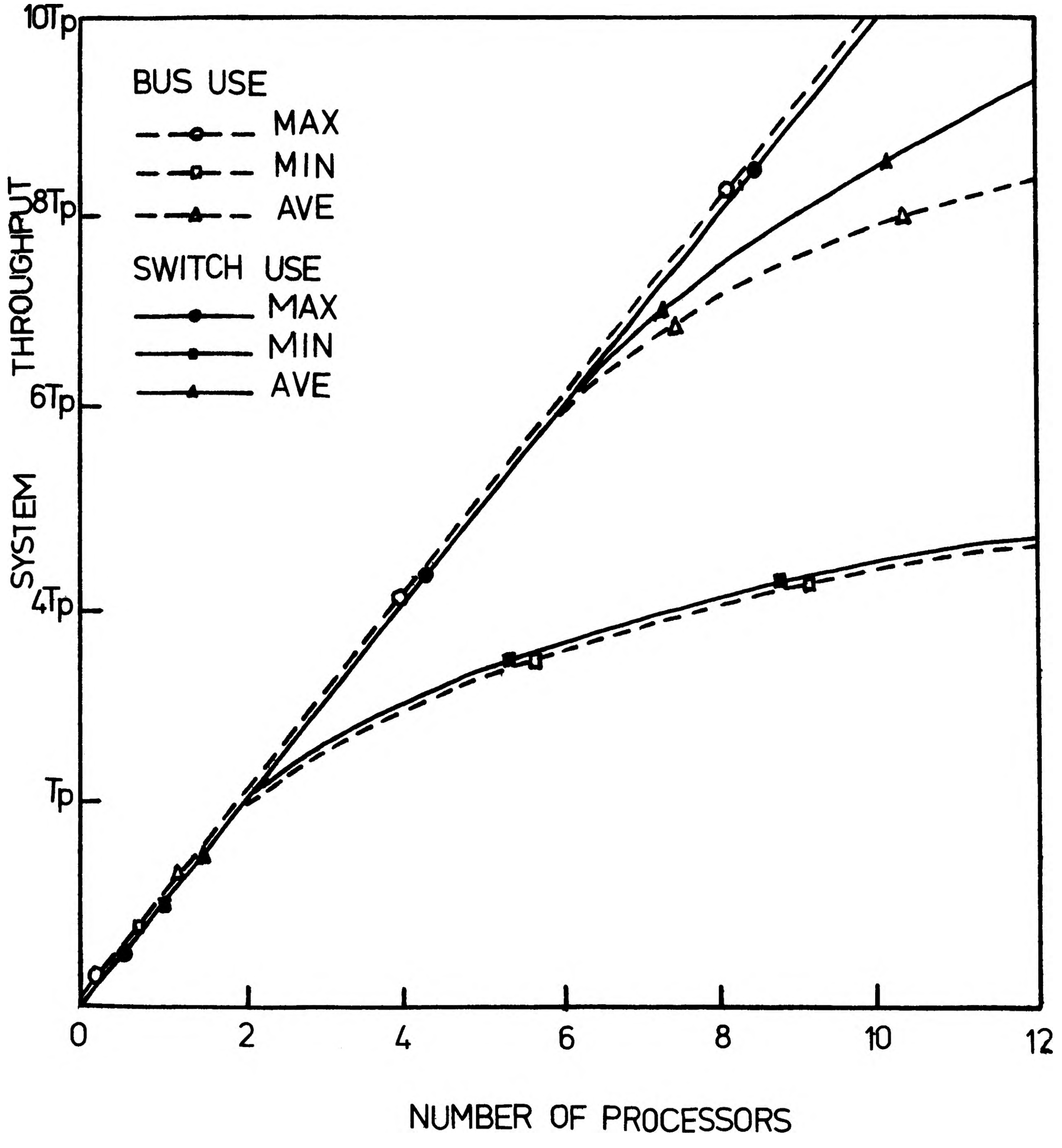

Figure 7. System Throughput Vs Number of Processors.

m=10

Figure 8 has been considered. The letters $I_1, I_2, \ldots, I_5$ denote the initial states, and the letters $F_1, F_2, \ldots, F_5$ denote the final states. The letter W denote the number of ways in which transition can occur. This number is the sum of different combinations to traverse the tree, e.g. the number of ways to reach state (3,1,0,0) from state (3,1,0,0) is (1x3+3x1) equal 6 ways. The matrix equation of the system considered has been derived as

$$\begin{bmatrix} P_4 \\ P_3 \\ P_2 \\ P_1 \\ P_0 \end{bmatrix} = \begin{bmatrix} 0.25 & 0.625 & 0.000 & 0.0156 & 0.0152 \\ 0.75 & 0.375 & 0.125 & 0.1875 & 0.1875 \\ 0.00 & 0.187 & 0.125 & 0.1406 & 0.1406 \\ 0.00 & 0.375 & 0.625 & 0.5625 & 0.5625 \\ 0.00 & 0.000 & 0.125 & 0.0937 & 0.0937 \end{bmatrix} \begin{bmatrix} P_4 \\ P_3 \\ P_2 \\ P_1 \\ P_0 \end{bmatrix}$$

with the constraint: $P_4+P_3+P_2+P_1+P_0 = 1$

The average number of busy memory modules is given by

$$\sum_{i=1}^{m} i\ p(i) = \sum_{i=1}^{m} i \sum_{j=1}^{m} p(j)\ A(i,j),$$

it follows that the average throughput will be given by

$$Ts\ (AVE) = Tp \sum_{i=1}^{m} i \sum_{j=1}^{m} P(j)\ A(i,j)$$

Table III shows the average number of busy memory modules for an 8x8 discrete Markov chain model during one cycle. This is in contradiction with the assumption made that the processing time is null. Figure 6 shows the average

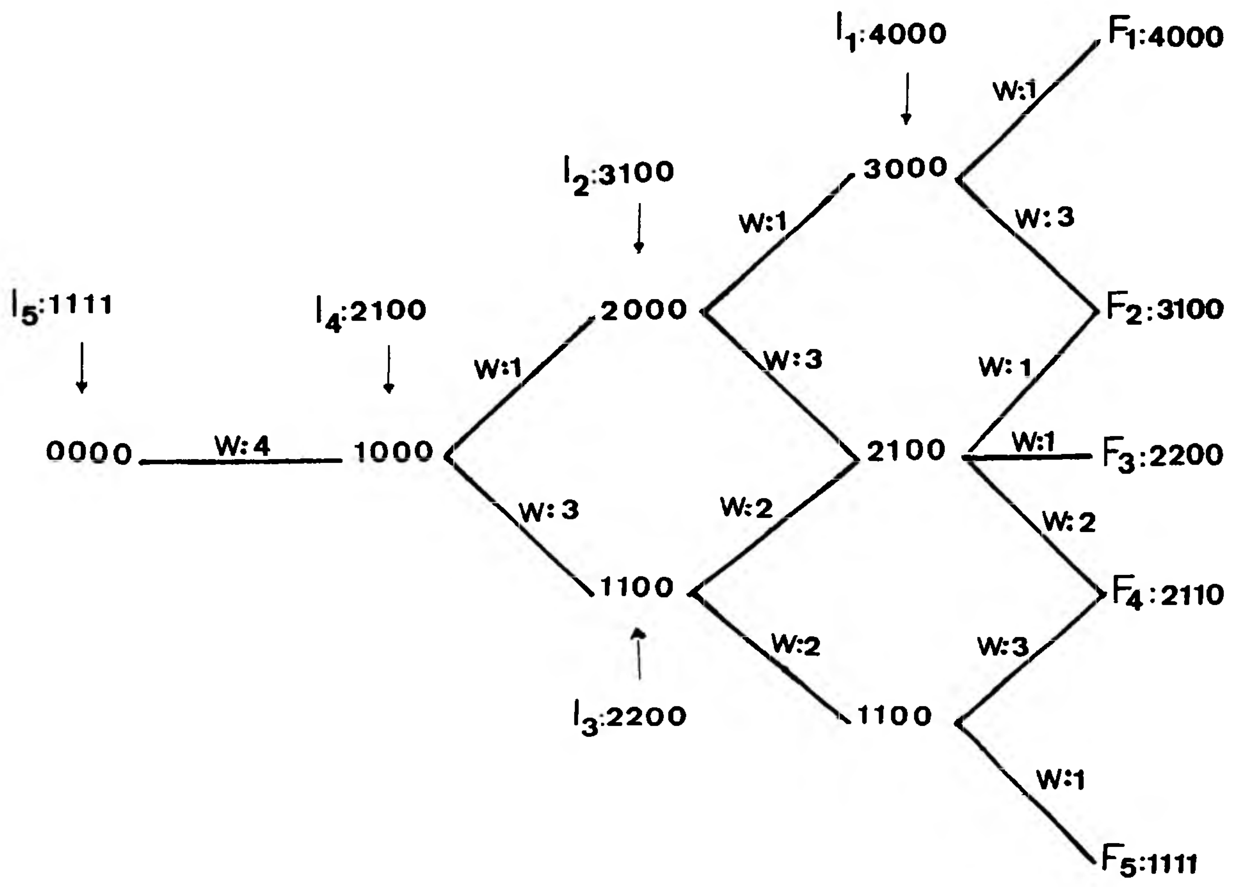

Figure 8. Transition States of a 4x4 System

TABLE III

DISCRETE MARKOV CHAIN MODEL

NUMBER OF PROCESSORS Pc = 1,2,...,8 (ROWS)

NUMBER OF MEMORY MODULES Mp = 1,2,...,8 (COLUMNS)

	1	2	3	4	5	6	7	8
1	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.0000	1.5000	1.6667	1.7500	1.8000	1.8333	1.8571	1.8750
3	1.0000	1.6667	2.0476	2.2692	2.4095	2.5054	2.5748	2.6272
4	1.0000	1.7500	2.2707	2.6210	2.8630	3.0365	3.1657	3.2652
5	1.0000	1.8000	2.4102	2.8633	3.1996	3.4530	3.6482	3.8019
6	1.0000	1.8333	2.5059	3.0370	3.4533	3.7809	4.0415	4.9471
7	1.0000	1.8571	2.5751	3.1663	3.6486	4.0418	4.3636	4.6292
8	1.0000	1.8750	2.6274	3.2657	3.9624	4.2521	4.6294	4.7491

throughput for a 3x3 system. Actually, the request generation rate follows a certain distribution. In order for the comparison of the present results with the time-shared bus performance results to hold, it is assumed that processors can generate new requests every cycle.

Feller [4] treating the "occupancy problem" had given the transition probability as

$$A^{(n)}(i,j) = \binom{m-i}{m-j} \sum_{v=0}^{j-i} \left(\frac{v+i}{m}\right)^{(n)} (-1)^{j-i-v} \binom{j-i}{v}$$

where $A^{(n)}(i,j)$ is the probability that there will be j occupied memories after n additional requests (cf. Appendix). Average throughput of the considered model has been plotted in Figure 7. Indeed, the plot shows that there is a more substantial increase of throughput with the number of processors than in the case of time-sharing configuration.

B. SYSTEM COST

In this section, the system cost shall be studied in order to determine how much the potential increase of the system due to the added throughput will cost. For this purpose the following subsystems costs have been defined:

Cr: Cost of system resources (including memory, mass storage, and peripheral devices).

Cp: Cost of an individual processor (including MOS LSI microprocessor chips, power supply cost, and mechanical assembly).

Cs: Cost of the switch (including wiring, control logic, arbitration and conflict solving, mechanical assembly of the switch).

For a system with p processors, the total system cost is derived as

$Ct = Cr + pCp + Cs$

Two systems have been considered: one in which $Cp=Cr/5$ and $Cs=P^2Cj$, the other in which $Cp=Cr/30$ and $Cs=p^2Ci$. Ci and Cj are the costs of individual switch node and its associated control and wiring, and have been equally chosen to be

$Ci=Cj=Cr/50$

Tp is assumed to be the same in both cases.

Figure 9 shows the increase in Ct with p. Another assumption that Cr is independent of the number of micro-processors p has been made. In reality, an increase in p may require an increase in total storage. As opposed to the results based on time-sharing technique, where cost of the system has been found to be linear with p, the cost of the present system using a switch has been found to be parabolic as expressed respectively by the equations of the two chosen systems:

$Ct=Cr\ (1+p/5+p^2/50)$ and $Ct=Cr\ (1+p/30+p^2/50)$

The information in Figures 7 and 9 has been combined in Figure 10 to indicate cost versus throughput. This cost of the system is a strong function of the ration Cr/Cp, and system cost increases rapidly as p approaches 10, diminishing the cost/effectiveness of the system. In order to determine the optimum number of microprocessors in the system, the

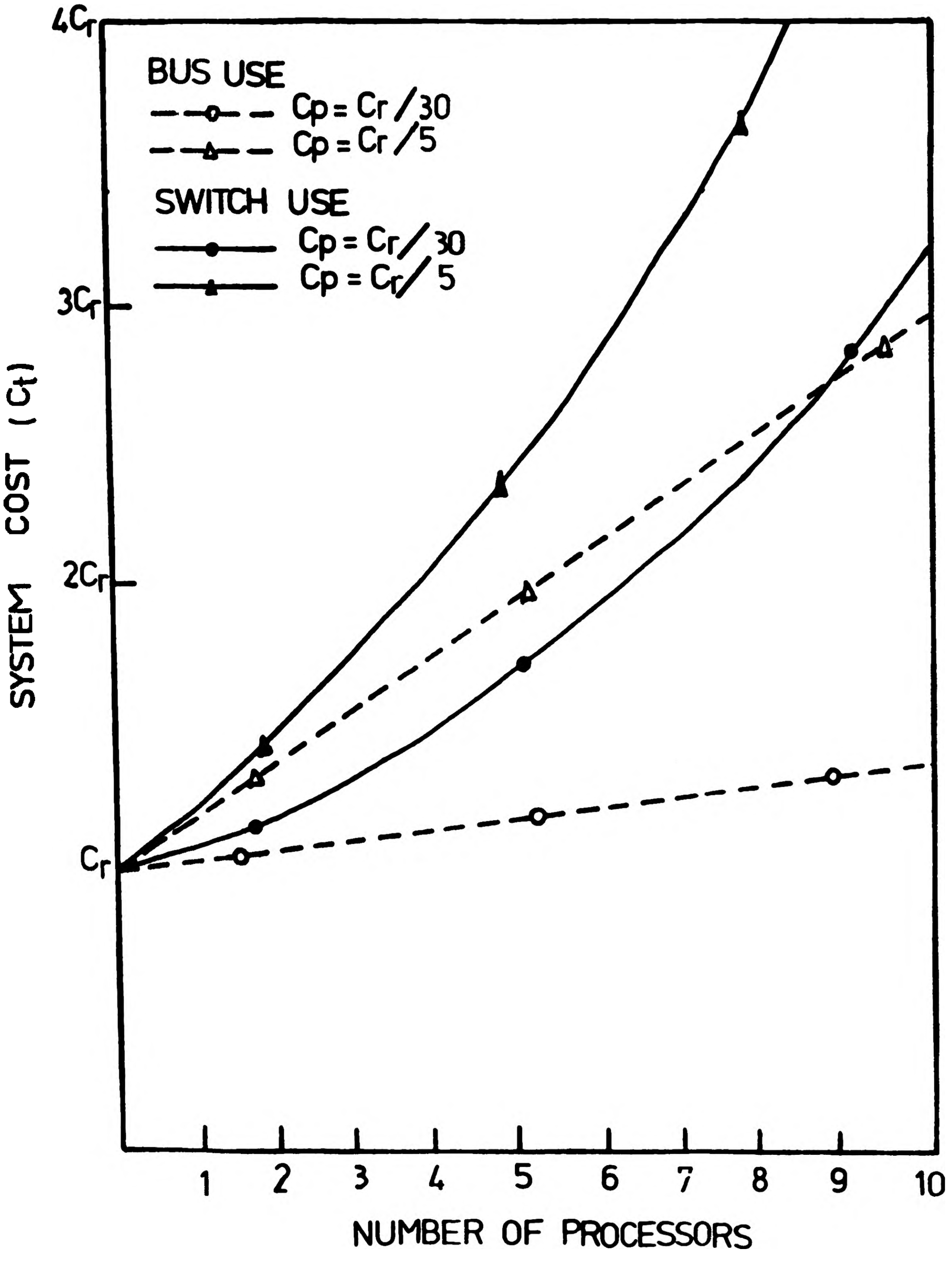

Figure 9. Relative System Cost Vs Number of Processors for Two Cases. m=10

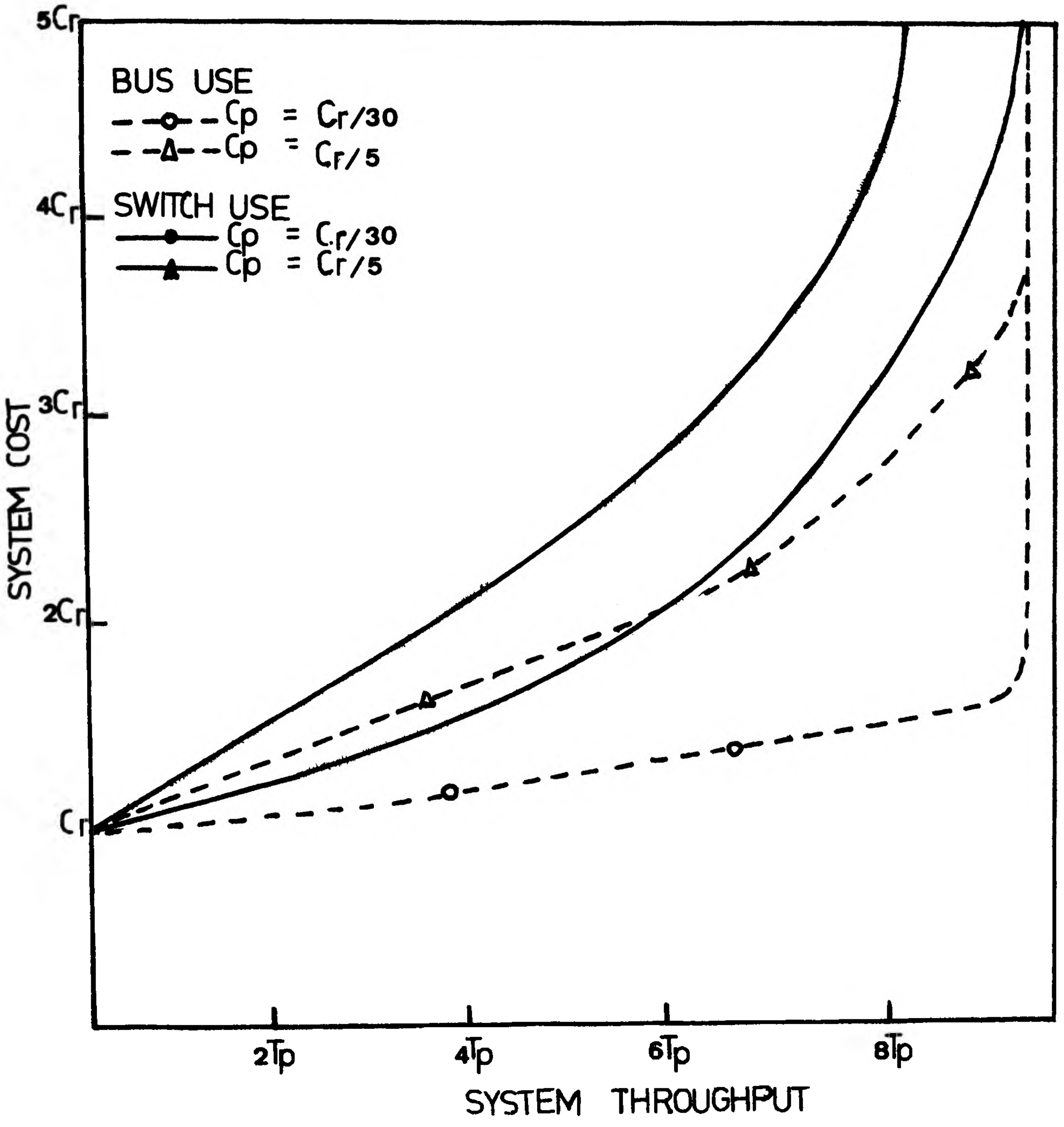

Figure 10. System Cost Vs System Throughput

m=10

ratio Ct/Ts has been calculated. This ratio is the cost per instruction execution that the user would have to pay. The information obtained from Figures 7 and 10, and plotted in Figure 11 show that, for two systems with analogous parameters the time system user will be paying less price per instruction execution than the space-shared system user as long as the number of processors in the system does not exceed 10. For a larger number, the space-shared configuration seems to be more attractive. This illustrates the advantages of minimizing both Cp/Cr and the value of m as shown.

For today's microprocessors, the ratio Cp to Cr is typically very low, since the cost of a complete microprocessor is in the range of several hundred dollars, while system memory and peripherals may be in the range of \$5,000 to \$20,000. However the cost of a switch increases as p^2. For the C.mmp computer developed at Carnegie-Mellon University, the cost of the switch turned out to be half the cost of the entire system. A means of decreasing the number of shared memory modules m in the system is to provide some memory local to each microprocessor. This approach has a double advantage of reducing memory reference interference and high access speed to data memory.

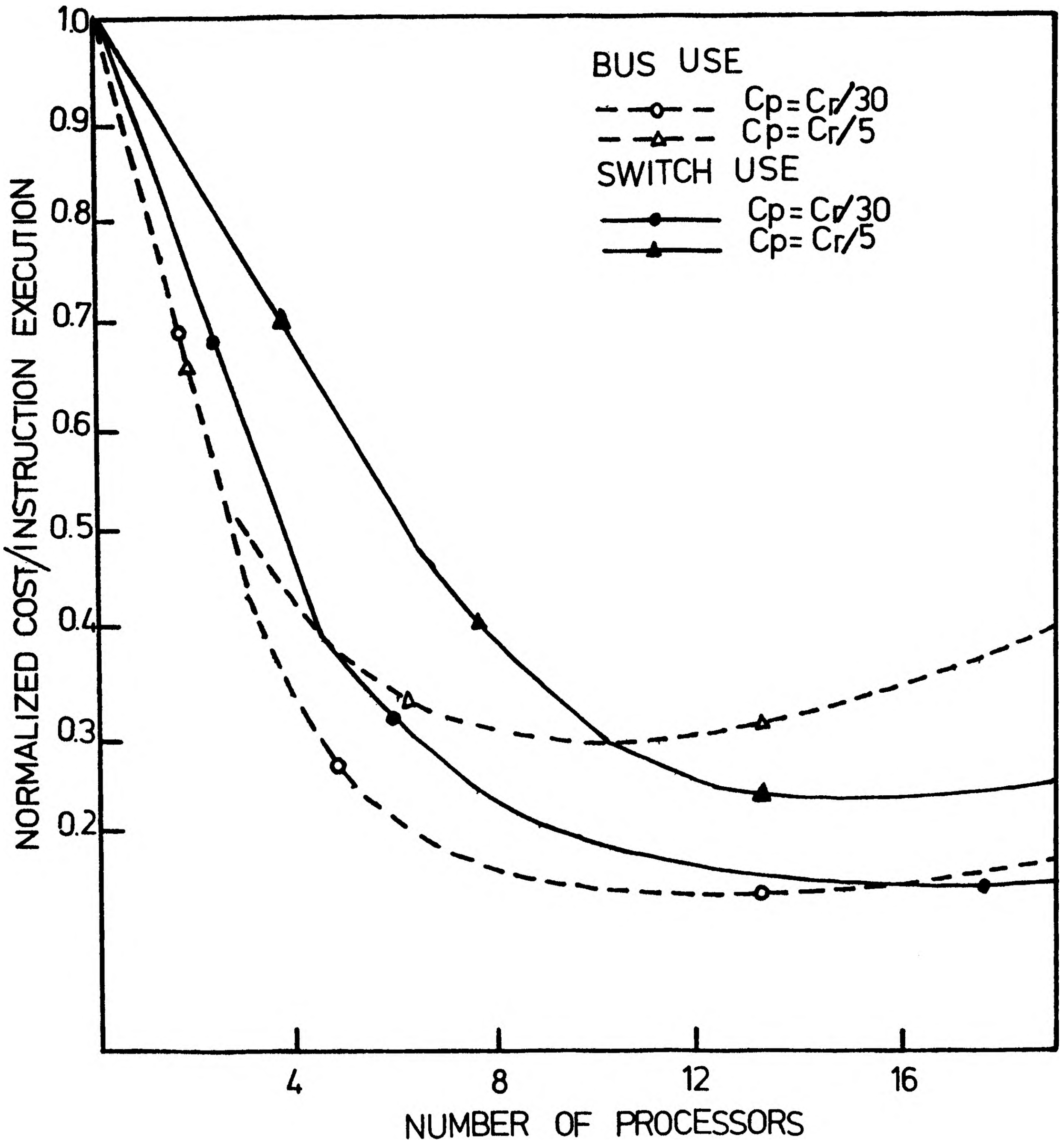

Figure 11. Normalized Curves of System Cost/Throughput Vs Number of Processors. m=10

IV. THE MIMD MACHINE

A. DEFINITION OF SIMD AND MIMD MACHINES

The example of multiprocessor systems chosen in this study was the MIMD type. Two types of parallel processing systems are single instruction stream-multiple data stream (SIMD) machines and multiple instructions stream-multiple data stream (MIMD) machines. An SIMD machine typically consists of a set of p processors and m memories, an interconnection network, and a control unit. The control unit broadcasts instructions to the processors and all active processors execute the same instruction at the same time. Thus a single stream instruction drives all the processors. Each processor executes instructions using data taken from a memory to which only it is connected. This provides a multiple data stream. The interconnection network allows interprocessor communications. A type of such a machine is the ILLIAC IV [5]. An MIMD machine typically consists of p processors and m memories, where each processor may follow an independent instruction stream. Hence, there are multiple data streams. As with SIMD there is a multiple data stream and an interconnection network. An example of such a machine is the C.mmp[6]. Figures 12 and 13 show the SIMD and MIMD computers respectively.

B. PARALLELISM THROUGH THE SWITCH NETWORK

A typical MIMD multiprocessor using a switch network as described previously has been considered. The first part

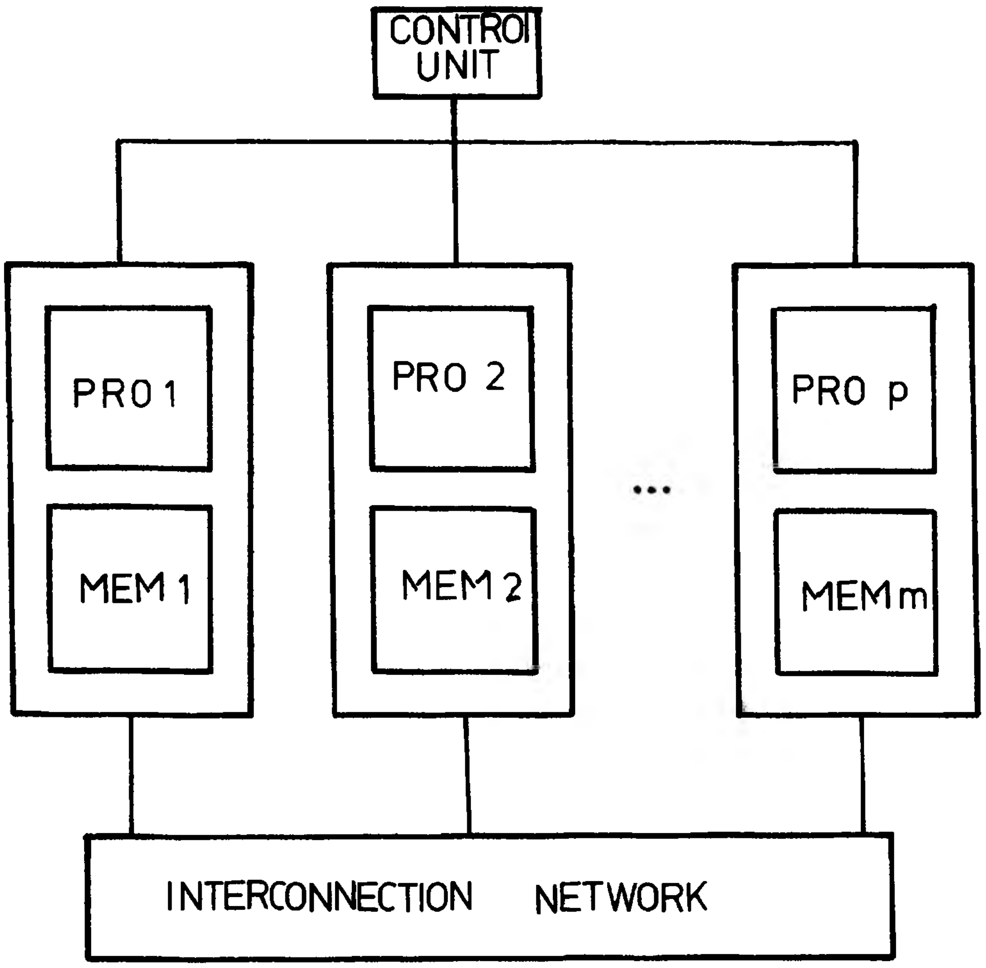

Figure 12. Block Diagram of a SIMD Computer

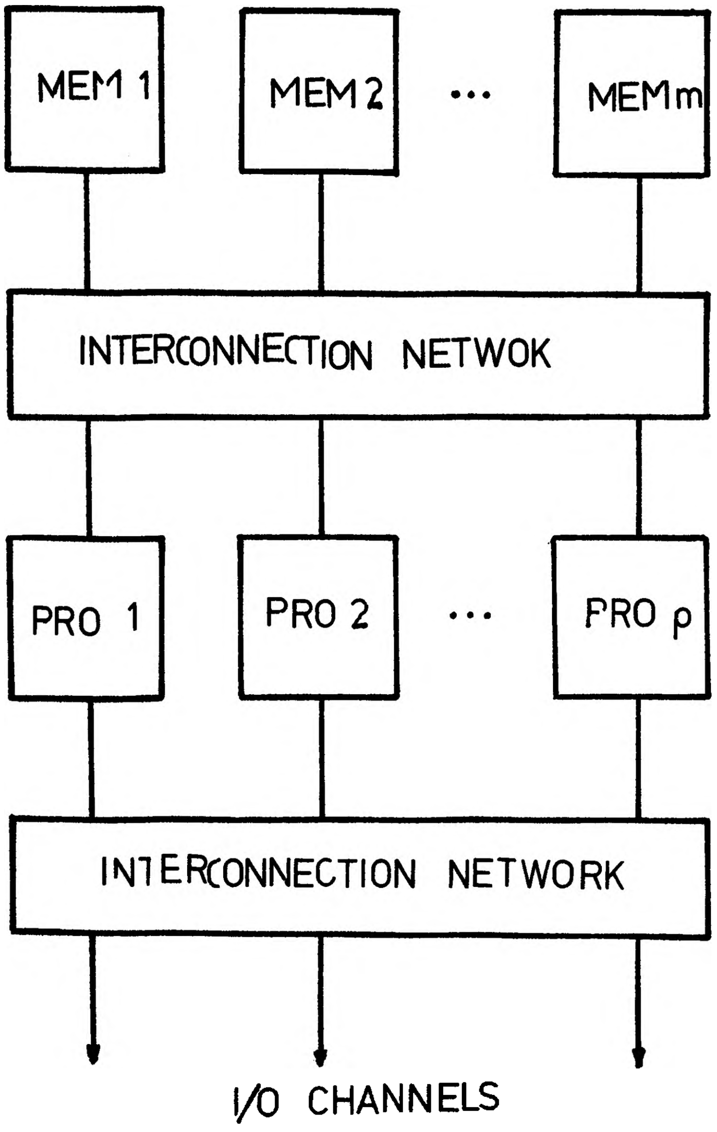

Figure 13. Block Diagram of a MIMD Computer

of this section will be devoted to the modeling of the switch node operation, then the rest of the section will examine problems related to two sensitive areas which are inter-processor control and resource sharing and scheduling. Petri nets appearing to be a clear and convenient way to express process coordination are used here to explore such problems. To avoid any ambiguity for the reader, the definition of Petri nets and the simulation rules are given explicitly.

1. Overview: James L. Peterson [7] defined a Petri net as in the following:

"A Petri net is an abstract, formal model of information flow. The properties, concepts, and techniques of Petri nets are being developed in a search for natural, simple and powerful methods for describing and analyzing the flow of information and control in systems that may exhibit asynchronous and concurrent activities."

Figure 14 shows a simple Petri net. The graph contains two types of nodes: Circles (called places) and bars (called transitions). The places and transitions are connected by direct arcs from places to transitions and from transitions to places. Place P_1 is an input to transition T_1 and places P_2 and P_3 are output to transition T_1. The execution of Petri nets is controlled by markers moving around the graph. Each place has one or more markers in it or may be empty. A transition is said to be enabled if all its input places contain at least one marker (or token). The transition fires by removing the enabling tokens from their input places

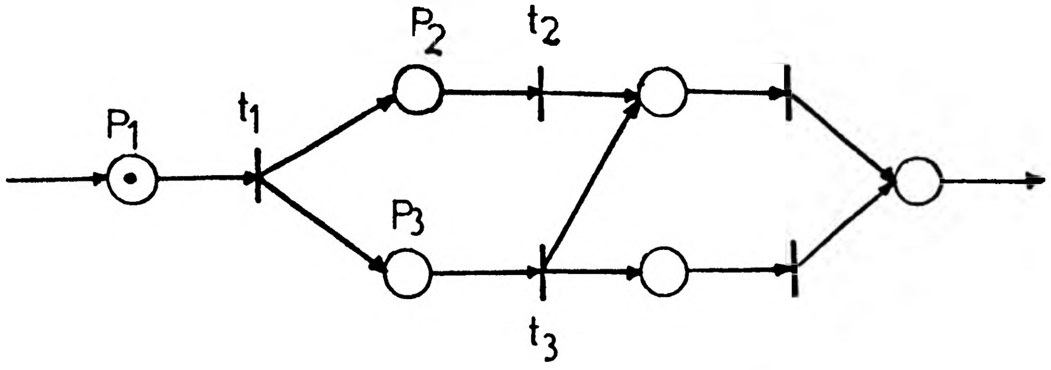

Figure 14. A Marked Petri Net

and generating new tokens which are deposited in the output places of the transition. Petri nets constitutes a broad area of study and the reader should consult the literature on this advanced theory for more information [8,9].

2. Node Switch Operation: When two requests arrive at a switch node, contention is essentially made as follows: the switch node selects one packet and rejects the other one if the two packets are to be passed to the same output. It takes time t_A to determine the successor node to which the message is to be sent. If that output is in use, it waits its turn for the use of the output link. When the selected output port becomes free, it takes time t_B for data to be available at the output port. Figure 15 models a timed Petri net of the switch node operation. Input A can select either output B or C, whereas input B can only select output C. Place P_5 cannot acquire a token and hence disables transition t_6 from firing. Consequently output D is forbidden to input B. When a message-packet is rejected by the switch node, it is automatically lost. If there is no conflict at the switch node level, processors carry out their tasks in a concurrent fashion, creating parallelism as it will be seen in the remaining of the section.

3. Interprocessor Control: A major concern about interprocessor control lies in synchronization of the processors to carry out a parallel computation correctly. To illustrate an example of this problem, the producer-consumer problem with one producer and two consumers is considered. The items

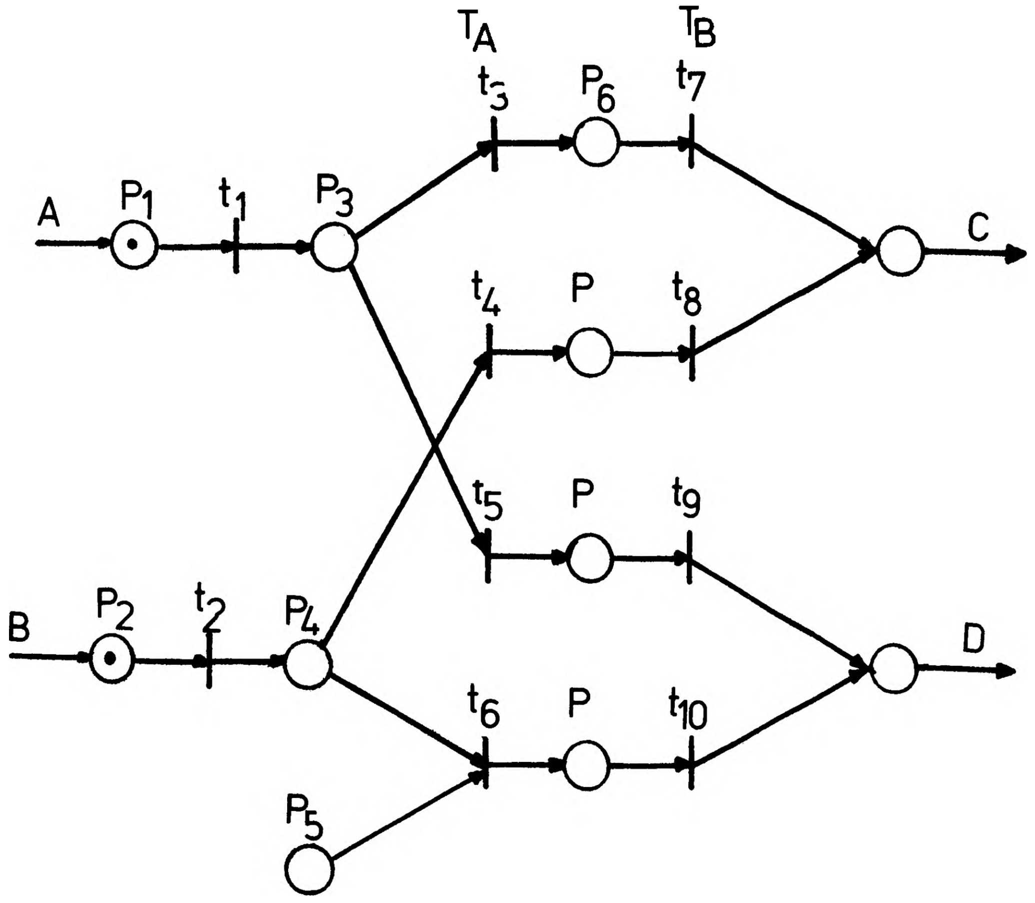

Figure 15. Switch Node Operation

produced by the producer are passed to the consumers to be picked up on a random basis. Only one item can be consumed by one consumer at a time. In order to avoid the access of the produced item by both consumers simultaneously, one of the consumers must lock the other from trying to consume the same item. The instruction to do this must be indivisible. The indivisibility can be achieved by instructions of the form "Test-and-Set" as implemented in many systems. Two portions of code generated by two processors wishing to access a common resource are called "critical sections". To control the correct execution of critical sections without conflict, Dijkstra [10] introduced a new concept using semaphores. A semaphore is a variable upon which a processor can execute a P and a V operation as in the following:

$V(S)$: $S \longleftarrow S + 1$

$P(S)$: L: If $S = 0$ then go to L else $S \longleftarrow S - 1$

Figure 16 summarizes the producer-consumer mutual exclusion protocol in terms of Petri nets. Places p_1 and p_2 represent the producer and places p_3, p_4, p_5 and p_6, p_7, p_8 represent the consumers. Place P_9 models the semaphore setting. Transitions t_1 and t_2 are mutually exclusive. Firing one disables the other automatically. Transitions t_3 and t_4 represent the critical sections of process 1 and process 2. Transitions t_1, t_2, t_5, t_6 control the entry and exit to the critical sections. Transition t_7 the production process. As can be seen, synchronization of concurrent processes can be achieved using semaphore techniques. Unfortunately, the

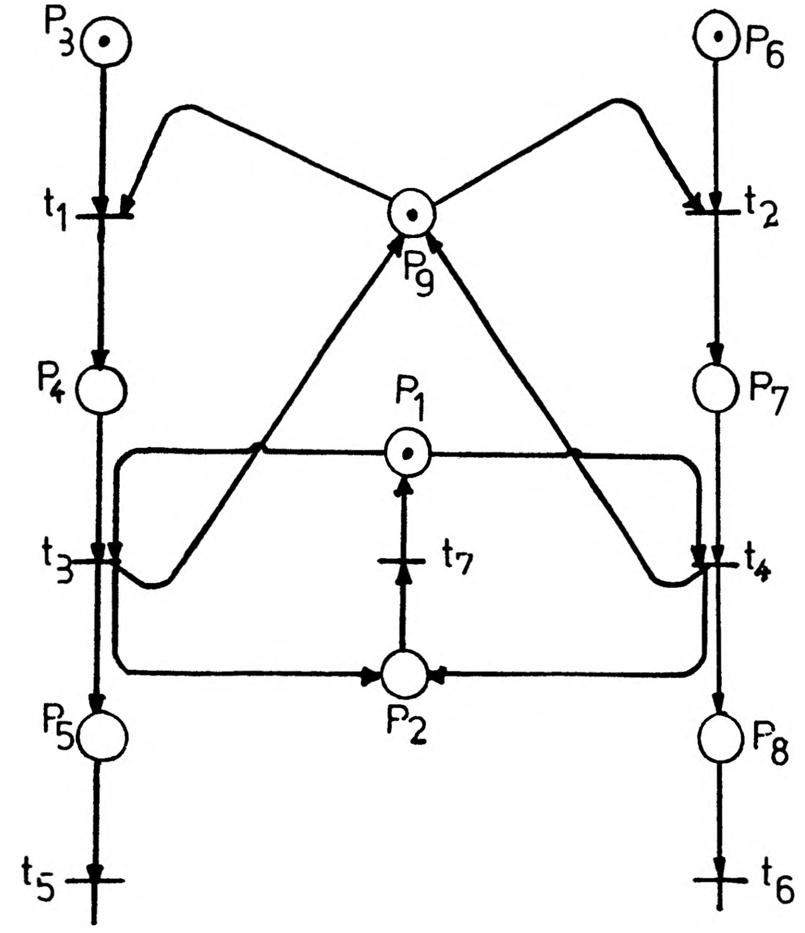

Figure 16. Illustration of the Producer-Consumer Problem

solution to these problems is related to scheduling and reliability problems. The failure of the processor whose process is in its critical section may lead to a dangerous situation. The rest of the processors will be blocked in an infinite testing loop. In the following discussion, a possible approach to the problem is explored. The use of a lock instead of a semaphore is provided. The lock consists of one part to test and set the lock and a busy signal bit to indicate that the processor executing the critical section code is successfully running. A process wishing to obtain the lock tests the appropriate part of the lock with a single indivisible instruction. If the result of the test indicates that the lock is free, it is then locked and the locking process can execute its critical code. If the lock part is set, the processor performs a second test upon the busy signal bit. If this bit is set then the processor using the resource is still executing properly. Otherwise, the operating system unlocks the lock indivisibly, allowing one of the waiting processors to proceed to use the lock and execute its critical code. Figure 17 shows a Petri net model of two concurrent processes using a lock. If either transition t_5 or t_6 does not fire, then the firing of transition t_2 or t_1 resets the lock at place p_7. Either process 1 or process 2 at place p_1 or p_4 has now a token in its place. Transition t_3 (or transition t_4) is now enabled and process 1 (or process 2) is now ready to use the lock and enter its critical section.

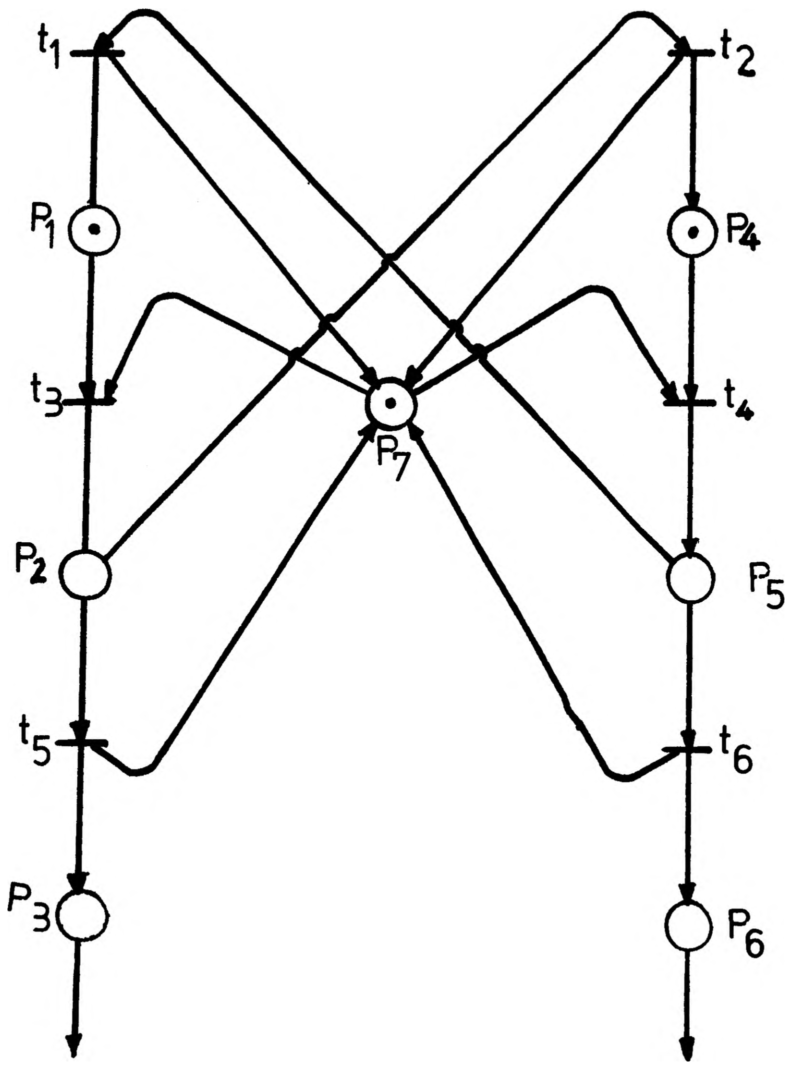

Figure 17. Solution to the Reliability Problem

4. Resource Sharing and Scheduling: In this section, the deadlock problem is examined. To illustrate the ideas once again, an example of a deadlock problem is considered. Two processes p_1 and p_2 request use of memory module M_1, M_2, M_3 as shown in Figure 18. Process p_1 acquires memory module M_1, M_3 and then needs M_2 and needs M_2, M_3 to carry on. The operating system's resource scheduler services process p_1's first request (transition t_1, M_1M_3) and process p_2's first request (transition t_2, M_2). From there, neither process can continue (the places p_1 and p_2 are empty and neither transition t_1, M_2 nor t_2, M_2 M_3 can fire).

To circumvent the deadlock problem, some prior conditions must be set before the system requirements are met. Prevention of system deadlock has been discussed and carefully analyzed in the literature [11]. In the light of this fruitful analysis and based on the conditions derived in order to avoid deadlock problems, a solution to the deadlock problem cited above is given and illustrated in terms of Petri net techniques as in Figure 19. The graph is self-explanatory. At transition t_3 and t_4, there is a mutual exclusion set by place p_3. Firing of either transition disables the other and enables its next transition in sequence by placing a token in either place p_4 or p_5 accordingly. This illustrates one of the conditions to avoid the deadlock problem which consists of preventing a process to hold exclusive control of some resources while a request for

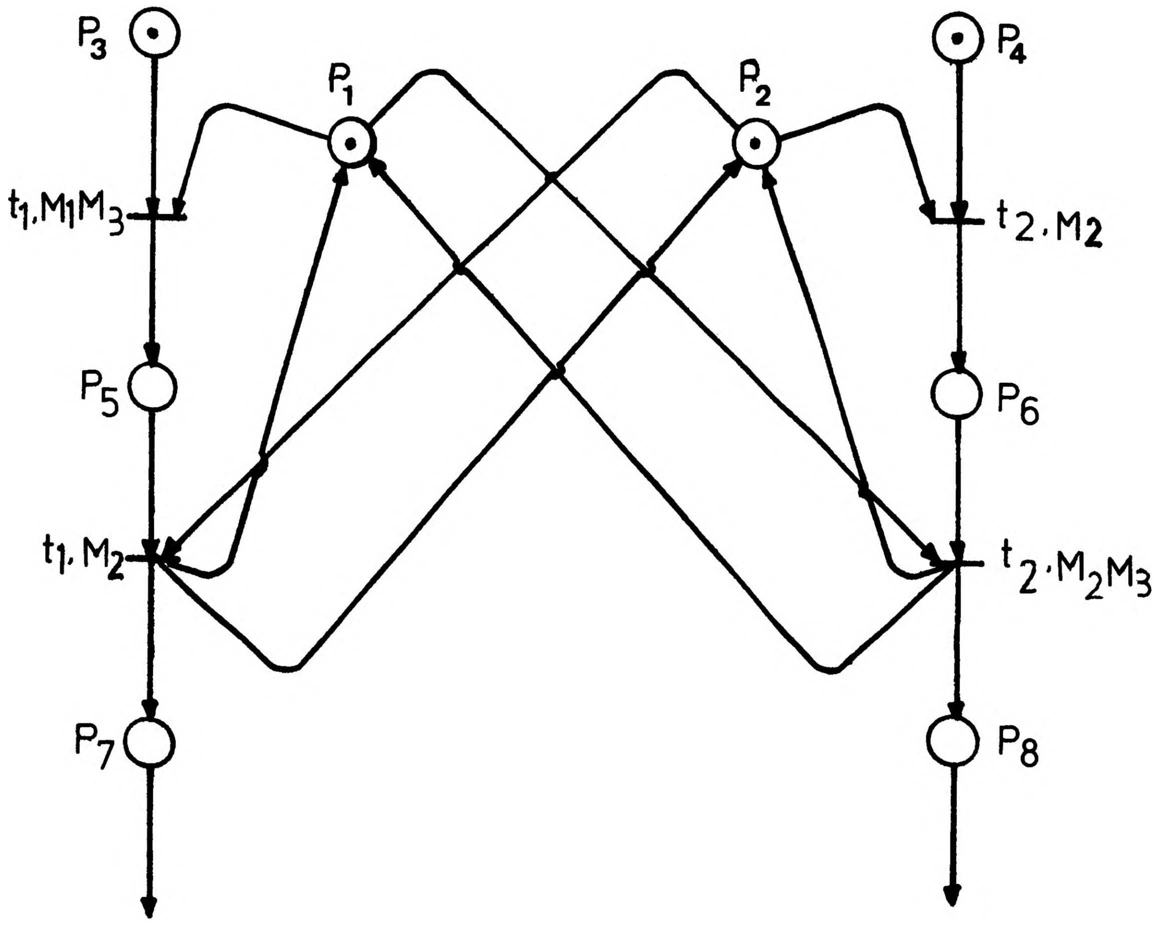

Figure 18. Illustration of The Deadlock Problem

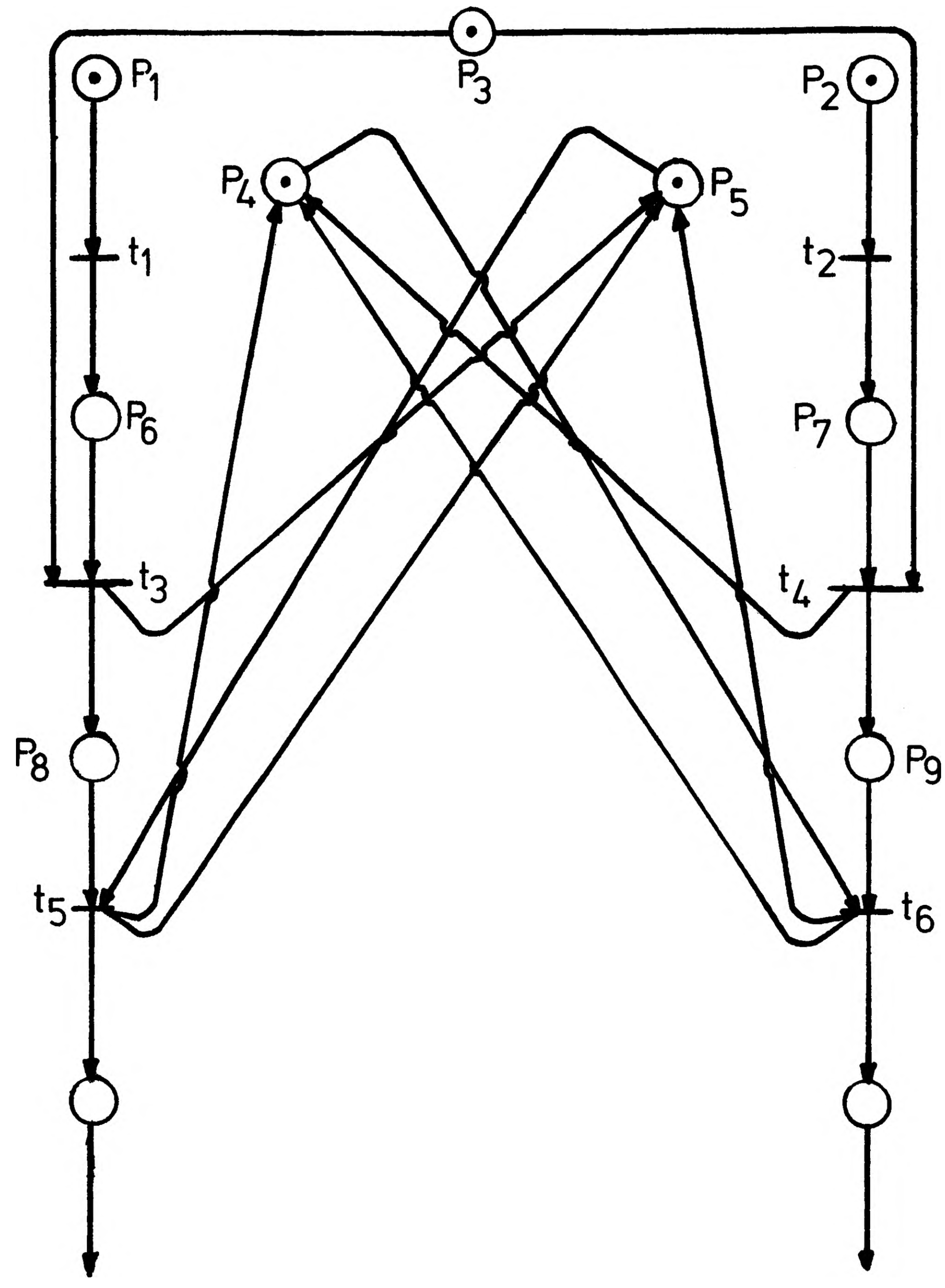

Figure 19. Solution to the Deadlock Problem

more resources is pending.

The deadlock problem has been a burden in the domain of multiprocessors task scheduling for years and the only way to circumvent it is to set preventing conditions which unfortunately increase operating system overhead.

V. CONCLUSION

It has been shown that a switch matrix configuration for the processor-memory interconnection network has reliability and expandability. If a switch node fails, the system can still function with less memory and degraded performance.

A computer system model has been used to estimate the relative performance of a computer using a switch network to another system using a bus. Performance bounds have been found to be equivalent in both systems. However, average throughput has been derived to be increasing more with the number of processors in the case of switch utilization. Certain simplifying assumptions have been made to make the analysis tractable but the model can be used to at least approximate the performance of some computer systems.

Expressions of the cost of the system have been given in the case where the cost of system resources C_r is thirty times and five times the cost of an individual processor C_p. It appears that in a certain number of processors range the space configuration handles large parallel computations better than the time configuration.

Parallelism through such a switch network has been viewed in terms of Petri net modeling techniques. The switch node operation has been investigated in detail.

Problems related to interprocessor control and resource sharing and scheduling have been studied. Possible approaches to their solutions have been given and validated through examples. When the switch node operation has been explored, it has been stated that a non-selected message-packet was rejected by the system and consequently lost.

A topic of significant importance would be the investigation of networks with buffering capability to permit request queueing and prevent this loss.

BIBLIOGRAPHY

1. Enslow, P.H. (Ed), Multiprocessors and Parallel Processing, John Wiley & Sons, N.Y. 1974

2. Reyling, G. Jr. Performance and Control of Multiple Microprocessors Systems, Computer Design, March 1974, pp 81-86.

3. Bhandarkar, D.P. Analysis of Memory Interference in Multiprocessors, IEEE Trans. on Comp. C-24 (Sep.1975) pp 897-908

4. Feller, W. An Introduction to Probability Theory and Its Applications. Vol. I, John Wiley & Sons, N.Y. 1968

5. Davis, R.L. The Illiac IV Processing Element, Trans.on Comp. C-18 (Sep. 1968), pp 800-816

6. Wulf, W.A. and C.G. Bell, C.mmp A Multimini Processor Proc. AFIPS 1972 Fall Joint Comp. Conf.41, AFIPS Press, Montvale, N.J. 1972, pp 765-777

7. Peterson, J. L. Petri nets, Computing Surveys, Vol.9 No.3 (Sep. 1977) pp 223-252

8. Murata, T. and Church,R.W. Analysis of Marked Graphs and Petri nets by Matrix Equations, Research Report MDC 1.1.8, Dept. Information Engineering, Uni.Illinois Chicago Circle (Nov. 1975) 25 pp

9. Petri, C.A. Concepts of net Theory in Proceedings Symp. and Summer School on Mathematical Foundation of Computer Science, High Tatras, (Sep. 1973) pp 137-146

10. Dijkstra, E.W., Solution of a Problem in Concurrent Programming, Comm, ACM 8, (Sep. 1965) pp.569-570.

11. Stone, H.S., Parallel Computers, Introduction to Computer Architecture, H.S. Stone, ed. SRA, Chicago, Ill. 1975, pp 318-374

VITA

Rabah Aoufi was born on March 2, 1955 in Medjana, Setif State, Algeria. He received his primary and secondary education in Medjana, Bordj-Bou-Arreridj, and Dellys (Algeria). In September 1974 he entered the University of Bab-Ezzouar, Algiers, which was opened to receive students for the first time. He received the equivalence of a Bachelor of Science Degree in Electrical Engineering from l'Ecole Polytechnique d'Alger in May 1977. He then came to the United States in January 1978 and attended an English course at Columbia University, New York. In September 1978 he entered the University of Missouri-Rolla and held the position of Graduate Teaching Assistant during the Fall of 1980.

APPENDIX

THE OCCUPANCY PROBLEM

Consider a sequence of independent trials, each consisting of placing a request at random at one of m given memory modules. The system is said to be in State p_k if exactly k memory modules are occupied. This determines a Markov chain with states $p_1, \ldots, p_m$ and transition probabilities such that

$$pjj = \frac{1}{m}, \quad pj,j + 1 = \frac{m-j}{m}$$

If p additional requests are placed at random, then $p_{jk}^{(p)}$ is the probability that there will be k busy and m-k free memory modules (so that $p_{jk}^{(p)} = 0$ if $k < j$).
the p step transition probabilities $p_{jk}^{(p)}$ is given by

$$p_{jk}^{(p)} = \sum_{r=j}^{k} \left(\frac{r}{m}\right)\binom{m}{r} \binom{r}{j} \binom{m-r}{k-r}(-1)^{k-r} \binom{m}{j}$$

on expressing the binomial coefficients in terms of factorials, this formula simplifies to

$$p_{jk}^{(p)} = \binom{m-j}{m-k} \sum_{V=0}^{k-j} \left(\frac{V+j}{m}\right)^{p} (-1)^{k-j-V} \binom{k-j}{V}$$

with $p_{jk}^{(p)} = 0 \qquad$ if $k < j$

(For a more specific demonstration of this Formula see [4])