ANALYTIC MODELS FOR MEMORY INTERFERENCE IN MULTIPROCESSOR COMPUTER SYSTEMS

Dileep P. Bhandarkar
Carnegie-Mellon University

Prepared for:
Air Force Office of Scientific Research
Advanced Research Projects Agency

September 1973
This thesis develops analytic models for estimating the amount of memory interference in multiprocessor systems, in which n processors access m memories independently. The processors are characterized by a typical processing time per memory access and the memories by an access time (ta) and rewrite time (tw). Processor behavior is simplified to an ordered sequence of a memory request followed by a certain amount of processing. The predominant technique used involves discrete time Markov chain models. Some simple exponential server models as well as several approximate models are also presented. Simulation...
Block 20. Abstract (Continued)

is used to evaluate the accuracy of the approximate models. Some empirical measurements of the PDP-11/20 are used to estimate the parameters of a model, that is used to predict the performance of C.mmp, Carnegie-Mellon University's multiprocessor computer, which will include up to 16 PDP-11 processors.
ANALYTIC MODELS
FOR
MEMORY INTERFERENCE
IN
MULTIPROCESSOR COMPUTER SYSTEMS

Dileep P. Bhandarkar

Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pa.
September 1973

Submitted to Carnegie-Mellon University
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This work was supported by the Advanced Research Projects Agency of the Office
of the Secretary of Defense (F44620-73-C-0074) and is monitored by the Air Force
Office of Scientific Research. This document has been approved for public
release and sale; its distribution is unlimited.
ABSTRACT

This thesis develops analytic models for estimating the amount of memory interference in multiprocessor systems, in which $n$ processors access $m$ memories independently.

The processors are characterized by a typical processing time per memory access and the memories by an access time($ta$) and rewrite time($tw$). Processor behavior is simplified to an ordered sequence of a memory request followed by a certain amount of processing. The predominant technique used involves discrete time Markov chain models. Some simple exponential server models as well as several approximate models are also presented. Simulation is used to evaluate the accuracy of the approximate models. Some empirical measurements of the PDP-11/20 are used to estimate the parameters of a model, that is used to predict the performance of C.mmp, Carnegie-Mellon University’s multiprocessor computer, which will include up to 16 PDP-11 processors.

The models can be partitioned into three broad classes: $tp=tw$, $tp>tw$ and $tp<tw$, where $tp$ denotes the average processing time. Systems with $tp=tw$ are described first because they represent boundary conditions for the other two cases. Different modeling techniques are examined for $tp=tw$ and a reasonable
approximation is proposed. An important result observed is the absence of a law of diminishing returns. The performance of a multiprocessor system with \( n \) processors and \( n \) memories continues to rise at a constant rate as \( n \) increases. A simple exponential server model showed this rate to be 0.5; a constant processing time model predicted a slope of 0.586 for the average number of busy Mp's. The exponential server model gives the average number of busy Mp's as \( \frac{n \cdot m}{n + m - 1} \). An approximate result for constant processing times gives the average number of busy Mp's as \( \frac{i \cdot j}{i - 1} \), where \( i = \max(n, m) \) and \( j = \min(n, m) \). An intuitively obvious conclusion limits the maximum number of active Pc's by \( \min(n, m) \).

Markov chain models are also developed for systems with \( t_p > t_w \). A new model for geometrically distributed processing time is developed. A different analytic approach is used to model systems with private caches for the processors. In general, since the Pc is slow, it takes fewer memory units for the performance to exhibit a saturation effect. In the absence of memory contention (which is now possible even for \( m < n \)) the maximum memory access rate \( (M_p A R) \) is \( n / (t_e + t_p) \).

With \( t_p < t_w \), since the processor is fast performance improvement is obtained for \( m > n \). If \( m = n \), these systems do not yield significant improvement over systems with \( t_p = t_w \). In general, adding an extra memory improves the performance more than adding an extra processor. The maximum average \( M_p A R \) is the minimum of \( m / t_e \) and \( n / (t_e + t_p) \); the maximum is achieved if the processors do not interfere. Note that since the Pc is very fast, it can make a request to the memory that served it last before the rewrite cycle is over. In this case, the Pc has to wait even though no other Pc is being serviced by the memory module.
I would like to thank Professor Samuel H. Fuller for his thoughtful guidance throughout the research reported in this thesis. I am also grateful to Professor J.W. McCredie Jr. for the many discussions with me during the early stages of the research. Acknowledgement is also due to Professors Bell, Grason and Siewiorek, who helped shape the final form of the dissertation. Since this thesis was printed on the Computer Science Department's Xerox Graphics Printer, everyone involved in the XCRIBL System effort deserves thanks. Finally, a special debt is owed to Ms. Lucile O'Donnell for her help and encouragement during the course of this research.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td></td>
<td>iii</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td></td>
<td>vi</td>
</tr>
<tr>
<td>Table of Contents</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>General Modeling Assumptions</td>
<td>6</td>
</tr>
<tr>
<td>1.2</td>
<td>Modeling Concepts</td>
<td>9</td>
</tr>
<tr>
<td>1.3</td>
<td>Extant Multiprocessor Systems</td>
<td>12</td>
</tr>
<tr>
<td>1.4</td>
<td>Comments on Earlier Work</td>
<td>14</td>
</tr>
<tr>
<td>1.5</td>
<td>Synopsis of Thesis Contents</td>
<td>17</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Multiprocessor Systems with (tp = tw)</td>
<td>21</td>
</tr>
<tr>
<td>2.1</td>
<td>Continuous Time Markov Chain Model</td>
<td>22</td>
</tr>
<tr>
<td>2.2</td>
<td>A Simple Discrete Markov Chain Model</td>
<td>25</td>
</tr>
<tr>
<td>2.3</td>
<td>Approximate Discrete Markov Chain Models</td>
<td>45</td>
</tr>
<tr>
<td>2.3.1</td>
<td>A New Approximate Discrete Markov Chain Model</td>
<td>45</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Strecker's Approximation</td>
<td>49</td>
</tr>
<tr>
<td>2.4</td>
<td>Discrete Markov Chain Model of Skinner and Asher</td>
<td>53</td>
</tr>
<tr>
<td>2.5</td>
<td>Diffusion Approximations</td>
<td>54</td>
</tr>
<tr>
<td>2.6</td>
<td>An Approximate Model for Arbitrary (P_{1/1})</td>
<td>55</td>
</tr>
<tr>
<td>2.7</td>
<td>Concluding Remarks</td>
<td>63</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Multiprocessor Systems with (tp &gt; tw)</td>
<td>68</td>
</tr>
<tr>
<td>3.1</td>
<td>Discrete Markov Chain Model for (tp = tw + tc)</td>
<td>68</td>
</tr>
<tr>
<td>3.1.1</td>
<td>General Technique for Constant (tp = tw + tc)</td>
<td>78</td>
</tr>
<tr>
<td>3.2</td>
<td>Discrete Markov Chain Model for Geometrically Distributed (tp)</td>
<td>81</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Extensions</td>
<td>87</td>
</tr>
<tr>
<td>3.3</td>
<td>McCredie's Exponential Server Model</td>
<td>91</td>
</tr>
<tr>
<td>3.4</td>
<td>Strecker's Analysis</td>
<td>93</td>
</tr>
<tr>
<td>3.5</td>
<td>The Effect of Cache on Systems with (tp &gt; tw)</td>
<td>95</td>
</tr>
<tr>
<td>3.6</td>
<td>Concluding Remarks</td>
<td>97</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Multiprocessor Systems with (tp &lt; tw)</td>
<td>100</td>
</tr>
<tr>
<td>4.1</td>
<td>An Approximate Model for (tp &lt; tw)</td>
<td>101</td>
</tr>
<tr>
<td>4.2</td>
<td>Strecker's Model</td>
<td>103</td>
</tr>
<tr>
<td>4.3</td>
<td>Concluding Remarks</td>
<td>106</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Empirical Measurements, Parameter Estimation and Model Validation</td>
<td>110</td>
</tr>
<tr>
<td>5.1</td>
<td>PDP-11/20 Overview</td>
<td>111</td>
</tr>
<tr>
<td>5.2</td>
<td>Validation of Unit Instruction Concept and Estimation of (tp)</td>
<td>116</td>
</tr>
<tr>
<td>5.3</td>
<td>Model Validation via C.mmp Performance Evaluation</td>
<td>125</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Conclusions</td>
<td>130</td>
</tr>
</tbody>
</table>
References

Appendix A Program Listings

Appendix B Description of Simulator

Tables

1.1 Salient Characteristics of Various Analytic Models

2.1 Some Properties of the Discrete Markov Chain Model
2.2 Comparison of Exact and Approximate Markov Chain Models
2.3 Expected Number of Busy Memories in One Cycle
2.4 Comparison of Simulation Results with Analytic Model of Sec. 2.3
2.5 Expected Number of Busy Memories in One Cycle
2.6 Expected Number of Busy Memories in One Cycle: Simulation Results
2.7 Characteristics of Various Models

3.1 Comparison of the Number of States
3.2 Transition Matrix for a 4x4 System with tp=tw+tc
3.3 Average Number of Busy Mp's: tp=tw+tc
3.4 Average Number of Busy Mp's: tp→Geometric Distribution

4.1 Average Number of Busy Mp's
4.2 Comparison of Analytic Models and Simulation Results

5.1a Direct Addressing Modes of PDP-11
5.1b Deferred or Indirect Addressing Modes of PDP-11
5.2 Effective Processing Time
5.3 Instruction Mix
5.4 Relative Frequency Distribution of the Effective Processing Time
5.5 Summary of Measurements on C.mmp
5.6 Analytic Results

Figures

1.1 A simple block diagram of a multiprocessor system
1.2a An example of the timing of a typical instruction
CHAPTER 1

INTRODUCTION

In the design of new computer systems there exists an enormous number of alternative decisions. In this thesis the major design parameters that will be allowed to vary are the number of processors (Pc's) and memories (Mp's) and their relative speeds. The quantitative approach to performance evaluation consists of three major phases [GrenU72]:

(i) Collection of data: this phase involves the planning and conducting of the experiment for data collection as well as techniques for measurement.
(ii) Analysis of data: this phase consists of construction of models and estimation of parameters in the models as well as validation of the models.
(iii) Interpretation of data: this phase concerns the summarization of the results and new insights gained in the study as well as making decisions based on the results.

The emphasis of the application of quantitative methods is on the convergence of two aspects. The first aspect is the reliance on data either from experimentation on the real system or from simulation. The second is the use of

†We use the PMS notation of Bell and Newell [BellC71a] in this thesis to describe hardware organization.
mathematical models. It is easy to collect massive amounts of confusing data unless one has some model. On the other hand, a model without empirical validation is at best an intellectual exercise.

Since no performance measurements of the actual system can be made until it has been designed, implemented, and then observed over a long period of time, it becomes necessary to use analytic and simulation models. Analytic models enable the designer to explore a large design space quickly and rather economically. However, modeling is not an easy task and it is often necessary to simplify the model to make it amenable to mathematical analysis, remembering that any mathematical model is only an approximation of real-life events. If the system is too complex to allow a complete analytic study, the system behavior can be modeled at various levels of abstraction in a hierarchical fashion.

Simulation offers an different approach: probabilistic emulation of a mathematical model that portrays the aggregate behavior of the real system. We gain in realism since we are no longer forced to impose assumptions for analytical convenience. However, simulations tend to be expensive if a high degree of realism on a detailed level is required. Due to the stochastic nature of simulation results, their precision can be measured by the standard deviation or confidence intervals of the estimates obtained. The confidence interval gets tighter as the size of the experiment is increased. The standard deviation is proportional to the square root of the length of the run. Hence, simulation studies are most valuable when focused on a small set of design alternatives.
Figure 1.1 A simple block diagram of a multiprocessor system.
selected by analytic studies. Also, if the analytic techniques are computationally expensive, simulation might well be more economical. Moreover, it is easier to change the mathematical model in a simulation experiment. Another important use for analytic models is as a control variable for improving the efficiency of simulation experiments by reducing the variance of parameter estimates from simulation experiments [GaveD71].

Mathematical models of computer systems can be developed at various levels of abstraction. A large number of models for time-sharing systems consider a job as a basic unit [McKlJ69], and in many models of multiprogrammed computer systems the block of instructions between I/O operations is taken as a basic unit [BuzeJ71; GaveD67]. However, in this study a much more detailed model is used to analyze interference as processors access individual words from the memory modules. Each processor's performance is measured by the number of memory accesses per unit time. In a multiprocessor system the performance of each Pc is not independent of the behavior of the other Pcs. A simple block diagram of a multiprocessor system is shown in Fig. 1.1. The connecting network or switch provides a path form each of the n Fc's to each of the m Mp's, such that a connection between Pc[i] and Mp[j] does not hamper a connection between Pc[k] and Mp[l], where i ≠ k and j ≠ l. The processors contend with each other for memory service. This contention is referred to as memory interference. This thesis presents a set of techniques for determining the extent of memory interference as measured by the average number of busy memories or the rate at which the Mp is accessed.
Legend:
1 instruction fetch
2 instruction decoding
3 operand fetch
4 instruction execution
5 next instruction fetch

Figure 1.2a: An example of the timing of a typical instruction.

Figure 1.2b: Simplified processor behavior: unit instruction. Two units model the instruction shown in Fig. 2.1a.
1.1 GENERAL MODELING ASSUMPTIONS

Due to the complexity of the problem, the exact detailed behavior of memory interference in a multiprocessor system is difficult to model. Some of the parameters that characterize the behavior of a PC are:

(i) Instruction mix: Instructions can be characterized by their relative frequency. In general, processor behavior varies for different instructions. However, in this thesis differences in instructions are not modeled explicitly. Processor behavior is modeled as an ordered sequence, consisting of a memory request followed by a certain amount of execution time. At this level of abstraction no distinction is made between the processing needed to decode an instruction and the processing corresponding to its execution. Thus, the processing time characterizing a PC depicts only the aggregate behavior of the real PC. Figure 1.2 depicts the actual and abstracted behaviors. A typical unit instruction† is shown in Fig. 1.2b.

(ii) Probability distribution of the processing time: Instructions are characterized by their processing time. Typical programs are measured to find the probability distribution of the instruction processing time.

(iii) Average processing time: This is obtained from measurements similar to those used for determining the probability distribution.

†The concept of an unit instruction was first proposed by Strecker[StreW70].
(iv) Access pattern of a Pc: This is the trace of the pages or memory locations accessed by the Pc. In this study serial correlation between successive memory accesses will be ignored; not a very serious assumption since data and instruction references are intermingled. Demand patterns will be modeled as sequences of Bernoulli trials. Memory accesses will be characterized by the memory unit to which they are addressed. Let \( p_{ij} \) denote the probability that the \( i \)-th processor requests service from the \( j \)-th memory unit. Thus, the demand pattern of each processor is equivalent to a sequence of Bernoulli trials. Unless otherwise specified, \( p_{ij} \) will be assumed to be equal to \( 1/m \), where \( m \) is the number of \( M_p \)'s.

The effect of I/O activity will not be modeled explicitly. Strecker [StreW70] has shown that if the rate of I/O requests is \( R_{IO} \), then a fraction \( R_{IO}/(m \cdot tc) \) of the memory access rate can be apportioned to I/O.

A processor is said to be queued if it is waiting for or in the process of receiving memory service. A processor is said to be active if it is currently being serviced by a memory. Likewise, a memory is said to be occupied or busy if there is at least one processor queued for that memory unit.

Primary memory behavior is a function of the fabrication technology, i.e. core or semiconductor. Memory performance can be characterized by the access time \( (t_a) \), rewrite time \( (t_w) \), and cycle time \( (t_c) \). Nominally, the cycle time is the sum of the other two. In this study, no distinction is made between read and
Figure 1.3 Structure of the queueing model
write operations. The effect of interleaving within a Mp module is to make the access and cycle times seen by a Pc variable. Most of the models in this thesis will use the average values.

The processing time shown in Fig. 1.2 is the effective processing time measured from the time when the Pc gets the data from its last memory access to the time when the next memory request reaches the memory. Thus, the delay associated with address mapping and communication protocol needed to make a request are attributed to the Pc. The memory access time includes the time required to set up the switch for the data transfer. If the memory control unit introduces some delay, then that is also added to the access time. A more detailed description is presented in Chapter 5.

1.2 MODELING CONCEPTS

A queueing model will be used to analyze memory interference. Figure 1.3 shows the basic structure of the model. All time delays are modeled as service centers and there is one job in the queueing system for every Pc. In real systems the Pc can start execution while the Mp is in its rewrite cycle. If $tp > tw$ the service time of the memory will be assumed to be $tc$, and the effective service time of the Pc will be assumed to be $tp - tw$. This simplification allows the Mp to start serving the next job as soon as the last job has left. The
overall system behavior is unaltered by this simplification. However, when \( t_p < t_w \)
the queueing model cannot be used for reasons explained in detail in Chapter 4.

The number of PC's will be denoted by \( n \) and the number of MP's by \( m \). The multiprocessor system will be referred to as a \( nxm \) system. The state of the system will be denoted by a vector describing the sizes of the various queues.

The major technique used in this thesis involves Markov chains [ParzE62]. A brief review of some of the definitions and concepts is presented here.

A stochastic process is a family of random variables \( X(t) \), \( t \in T \) indexed by a parameter \( t \) varying in an index set \( T \). The stochastic process is a discrete parameter process if \( T = \{0,1,2,\ldots\} \) or \( \{0,\pm1,\pm2,\ldots\} \). The process is a continuous parameter process if \( T = \{t \geq 0\} \) or \( \{-\infty < t < \infty\} \).

A discrete parameter stochastic process \( \{X(t), t = 0,1,2,\ldots\} \) or a continuous parameter stochastic process \( \{X(t), t \geq 0\} \) is said to be a Markov process if, for any set of \( n \) points \( t_1, t_2, \ldots, t_n \) in the index set of the process, the conditional distribution of \( X(t_n) \), for given values of \( X(t_1), \ldots, X(t_{n-1}) \), depends only on \( X(t_{n-1}) \), the most recent known value; more precisely, for any numbers \( x_1, \ldots, x_n \)

\[
P[X(t_n) \leq x_n \mid X(t_1) = x_1, \ldots, X(t_{n-1}) = x_{n-1}] = P[X(t_n) \leq x_n \mid X(t_{n-1}) = x_{n-1}]
\]

Intuitively, this means that, given the present of the process, the future is independent of its past. The set of possible values of a stochastic process is called its state space. The state space is called discrete if it contains a
finite or countably infinite number of states. A state space which is not discrete is called continuous. A Markov process whose state space is discrete is called a Markov chain.

A Markov process is described by a transition probability function, denoted by \( P(E,t \mid x,t_0) \), which represents the conditional probability that the state of the system will at time \( t \) belong to the set \( E \), given that at time \( t_0 (\leq t) \) the system is in state \( x \). The Markov process is said to have stationary transition probabilities, or to be homogeneous in time, if \( P(E,t \mid x,t_0) \) depends on \( t \) and \( t_0 \) only through the difference \( (t-t_0) \).

A Markov chain is irreducible if every state can be reached from every other state not necessarily in one step. The period of a state \( i \) is defined as the greatest common divisor of all integers \( k \) such that the probability of returning to state \( i \) in \( k \) steps is greater than 0. A state of an irreducible Markov chain is aperiodic if it has period 1. A Markov chain is aperiodic if every state in its state space is aperiodic.

A discrete parameter irreducible aperiodic Markov chain that has stationary transition probabilities possesses a stationary state probability distribution. Let \( Z(k) \) denote the steady state probability of state \( k \). Then,

\[
Z(k) = \sum \text{TRANS}(k,j) \times Z(j)
\]

where \( \text{TRANS}(k,j) \) is the one step transition probability from state \( j \) to state \( k \).

Transition probabilities from a current state to a next state will be
evaluated for the irreducible aperiodic discrete Markov chain models in the forthcoming chapters. The steady state probabilities will be used to calculate the average number of busy Mp’s which is equal to the number of unit instructions executed in one memory cycle. The *unit instruction execution rate* (UER) or *memory access rate* (MPAR) is obtained by dividing the average number of busy Mp’s by the cycle time.

### 1.3 EXTANT MULTIPROCESSOR SYSTEMS

A group at Carnegie-Mellon University is currently in the process of constructing a multiprocessor computer system (C.mmp) that will have up to sixteen central processors (PDP-11/20’s) sharing the same physical address space [BellC71b; WulfW72] and concern has been expressed about the performance of such a system with this many active processors. The models developed in this thesis will be used to predict the performance of C.mmp in Chapter 5. Figure 1.4 illustrates the major components of a multiprocessor such as C.mmp. In addition to the processors, there is a set of memory modules that are able to operate independently; little would be gained if all the processors had to wait for service from a single memory module. Thus, between the processors and the memory modules (Mp’s) is an *n* by *m* crosspoint switch, which allows any Pc to access any Mp. There are a number of ways of implementing the switch; Fig. 1.5(a) depicts
where: Pc/central processor; Mp/primary memory; T/terminal;
Ks/slow device control (e.g., for Teletype);
Kf/fast device control (e.g., for disk);
Kc/control for clock, timer, interprocessor communication
Dmap/relocation registers for mapping Pc address into Mp address space.

Both switches have static configuration control by manual and
program control

Fig. 1.4 Proposed CMU multiminiprocessor computer/C.mmp.
1.3 Extant Multiprocessor Systems

an n by m crosspoint switch, and Fig. 1.5(b) illustrates the use of trunk lines; combinations of these two basic schemes can yield many other schemes. Other multiprocessors, although limited to a small number of PC’s, i.e. two to four, also basically use a crosspoint switch, e.g. the Burroughs D825[AndeJ62] and Univac 1110. For further discussion of trunk lines, and a variety of other switching structures, the reader is referred to Bell and Newell [BellC71a].

1.4 COMMENTS ON EARLIER WORK

A review of current literature shows very few models of memory interference. Skinner and Asher [SkinC69] proposed a discrete Markov chain model for multiprocessor systems with tp=tw. The analysis was presented for a small number of PC’s(<2). However, for larger systems the complexity of the problem deterred the authors from further pursuit of an analytic solution.

Strecker[StreW70] developed a set of simple approximate models. Most of his modeling assumptions are similar to those used in this thesis. While the analysis of Skinner and Asher is rigorous and exact, Strecker’s analysis is approximate. In this thesis, an exact analysis of a discrete Markov chain model for systems with tp=tw is presented. As expected, the exact analysis is very complex. However, the results of the exact analysis suggest more reasonable approximations that yield performance estimates that are more accurate than
Figure 1.5a m\times n crossbar switch

Figure 1.5b k-trunk line switch
Chapter 1: Introduction

1.4 Comments on Earlier Work

Strecker's. For instance, the exact discrete Markov chain model described in Chapter 2 shows that the $M_p|AR$ for a $j \times k$ and a $k \times j$ multiprocessor system with $t_p=t_w$ is almost equal, a result not apparent from Strecker's work. Also, Strecker's formula for $t_p=t_w$ is more accurate for $m>n$ than for $m<n$; $n$ is the number of Pc's and $m$ the number of Mp's.

More detailed descriptions of the works of Skinner and Asher, and Strecker can be found in Chapters 2, 3 and 4. Bhatia[872] has shown how the results from memory interference models can be used as data for models of timeshared multiprocessor systems at the user program level.

A major contribution of this thesis is a systematic approach to the use of the Markov chain technique for analyzing memory interference in multiprocessor systems. The exact analysis of the Markov chain is complex. However, the behavior observed from the exact analysis is used to examine an approximate solution technique that is computationally simpler. Though some of the models presented in this thesis may be only marginally more accurate(5%) than Strecker's results, they may result in much more accurate estimates when used as inputs to other models such as Bhatia's model for time-shared systems. For example, the waiting time for a single server queueing system with Poisson input rate $\lambda$ and exponential service at rate $u$ is $1/(u-\lambda)$. A 5% error in the value of $u$ can cause a greater error in the estimated waiting time if $\lambda$ is close to $u$. 
1.5 SYNOPSIS OF THESIS CONTENTS

The analytic models are described in detail in Chapters 2, 3 and 4. The models are mathematical abstractions of the real systems. Thus, they do not exactly reflect the true behavior of the physical system. However, the models will be referred to as exact or approximate depending on the quality of the technique used to analyze the mathematical model.

The models can be grouped into three broad classes: $tp=tw$, $tp>tw$ and $tp<tw$, where $tp$ denotes the average processing time. Systems with $tp=tw$ are described first because they represent boundary conditions for the other two cases. Different modeling techniques are examined for $tp=tw$ and a reasonable approximation is proposed. A casual reader may find it useful to glance through the empirical results and validation of the models presented in Chapter 5 and the concluding remarks summarized in Chapter 6 before examining the mathematical intricacies of Chapters 2, 3 and 4. A more detailed summary of the thesis content is given below. Table 1.1 summarizes the salient characteristics of the various analytic models.

Chapter 2 is devoted to multiprocessor systems with $tp=tw$. A simple exponential server model provides some insight into the effect of adding a processor or a memory to the system. A more elaborate analysis for constant processing time uses discrete Markov chain techniques; an exact but unwieldy
### TABLE 1.1

<table>
<thead>
<tr>
<th>Model Descriptor</th>
<th>Processing Time</th>
<th>Memory Cycle Time</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPROCESSOR SYSTEMS WITH ( tp &gt; tw )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Continuous Time Markov Chain</td>
<td>exponential</td>
<td>exponential</td>
<td>Jackson's Formulae are used to obtain a simple closed form solution.</td>
</tr>
<tr>
<td>Exact Discrete Markov Chain</td>
<td>constant</td>
<td>constant</td>
<td>The solution is algorithmic. Unwieldy for large systems.</td>
</tr>
<tr>
<td>Approx. Discrete Markov Chain</td>
<td>constant</td>
<td>constant</td>
<td>Approximation: non-active ( P_c )'s are reassigned to busy ( M_p )'s at the end of the cycle. Good for ( n &gt; 3m ).</td>
</tr>
<tr>
<td>Strecker's Approximation</td>
<td>constant</td>
<td>constant</td>
<td>Simple closed form solution. Less accurate than above. Non-active ( P_c )'s are reassigned to all ( M_p )'s.</td>
</tr>
<tr>
<td>Skinner and Asher's Discrete Markov Chain</td>
<td>constant</td>
<td>constant</td>
<td>Exact discrete Markov chain analysis for upto 2 ( P_c )'s and ( m ) ( M_p )'s.</td>
</tr>
<tr>
<td>Approximate Model for Arbitrary ( P_j )</td>
<td>constant</td>
<td>constant</td>
<td>( P_j ) is not restricted to ( 1/m ). Solution is simple but approximate.</td>
</tr>
<tr>
<td>MULTIPROCESSOR SYSTEMS WITH ( tp &lt; tw )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Discrete Markov Chain for ( tp &lt; tw + \epsilon )</td>
<td>constant</td>
<td>constant</td>
<td>Approximate analysis is presented. Queued ( P_c )'s are reassigned to all ( M_p )'s at the end of the cycle.</td>
</tr>
<tr>
<td>Discrete Markov Chain for Geometrically Distributed ( tp )</td>
<td>geometric</td>
<td>constant</td>
<td>Approximate analysis. [ \text{Prob}(tp &lt; tw + \epsilon) = \frac{\epsilon}{\lambda} ]</td>
</tr>
<tr>
<td>McCredie's Exponential Server</td>
<td>exponential</td>
<td>exponential</td>
<td>Jackson's formulae are used. One ( M_p ) has different ( t_c ) and different access probability. Cache.</td>
</tr>
<tr>
<td>Strecker's Analysis</td>
<td>constant</td>
<td>constant</td>
<td>Little's Formula is used.</td>
</tr>
<tr>
<td>Approximate Model for Systems with cache</td>
<td>constant</td>
<td>constant</td>
<td>Little's Formula is used. Cache memory speed is a parameter.</td>
</tr>
<tr>
<td>MULTIPROCESSOR SYSTEMS WITH ( tp = tw )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>An Approximate Model</td>
<td>constant</td>
<td>constant</td>
<td>Model for ( tp = tw ) is used to obtain the conditional probability of ( P_c )'s second request going to an idle ( M_p ), depending on the number of busy ( M_p )'s.</td>
</tr>
<tr>
<td>Strecker's Approximation</td>
<td>constant</td>
<td>constant</td>
<td>Results of ( tp = tw ) are used to find the probability of request to an idle ( M_p ). Average number of busy ( M_p )'s is used.</td>
</tr>
</tbody>
</table>
Chapter 1: Introduction
1.5 Synopsis of Thesis Contents

analysis and a simple approximate analysis is presented. This exact analysis of the Markov chain model is compared with Strecker's approximation. A new approximate model is introduced to analyze the effect of skewing the access patterns of the processors so that each has a greater preference for a different memory module.

Chapter 3 presents discrete Markov chain models for \( t_p > t_w \). Techniques for an exact analysis of the models are introduced and some approximations suggested. Models are developed for constant processing time. McCredie's exponential server model\([Mccr73]\) and Strecker's approximate model for constant \( t_p \) are discussed. Two new models for analyzing the effect of cache memories are also described.

Chapter 4 contains an approximate model for \( t_p < t_w \) and compares the results with Strecker's model.

Chapter 5 contains the results of some empirical measurements of PDP-11 programs. The processing time distribution is evaluated from these measurements. The process of extracting the abstract model parameters from the real physical system behavior is demonstrated. Predictions are made about the performance of Carnegie-Mellon University's C.mmp and compared with some actual measurements. This preliminary comparison with actual measurements shows the accuracy and utility of analytic models.

Chapter 6 summarizes the salient results developed in the thesis. An
Chapter 1: Introduction
1.5 Synopsis of Thesis Contents

example of the use of these models for examining design alternatives is also included. Some directions for future work are discussed.
In this chapter multiprocessor systems with $tp=tw$ will be analysed. This could happen even with a very fast $Pc$. If the system is bus-bound and the $Pc-Mp$ bus recovers at the same time that the memory is ready to service the next request, then the effective processing time (as seen by the memory) is equal to the memory rewrite time. With $tp=tw$, the analysis is simpler than with $tp<tw$ and $tp>tw$. Also, it is a boundary condition for the other two cases. Thus, $tp=tw$ is an interesting case for a preliminary comparison of various modeling techniques, even when $tp$ is not equal to $tw$ in reality.

A simple exponential server model provides some insight into the effect of adding a processor or a memory to the system. A more elaborate analysis for constant processing time uses discrete Markov chain techniques; an exact but unwieldy analysis and a simple approximate analysis is presented. This exact analysis of the Markov chain model is compared with Strecker's approximation. The results of the exact analysis are used to improve the accuracy of the approximate analysis. A new approximate model is introduced to analyze the effect of skewing the access patterns of the processors so that each has a greater preference for a different memory module. A diffusion approximation is also considered.
2.1 CONTINUOUS TIME MARKOV CHAIN MODEL

In our first model, we apply the classic simplifying assumption in queueing models: we model the service time, or cycle time, of the memory modules as exponentially distributed random variables [cf. WagnH69]. Clearly most memory systems do not have an exponentially distributed cycle time. However, techniques such as interleaving, cache memories, and the type of memory access (read, write, read-modify-write) suggest that this exponential assumption may be as good an approximation as the assumption that the memory cycle time is fixed, and not variable at all. Without further assumptions or approximations, we can use the results of Jackson [JackJ63] and Gordon and Newell [GordW67] to find the performance of the multiprocessor system. This technique is also used by McCredie [McCrJ73] for multiprocessors with tp>tw. This exponential server model is the simplest model to analyze. It also gives some basic insight into the extent of memory interference when the system has a large number of PC's and MP's.

Let the number of service centers be m. The states of the system are m-dimensional vectors with non-negative integer components, the j-th component representing the queue length at center j. If \( K=(k_1, k_2, \ldots, k_m) \) is a state vector, then let \( S(K) = \sum_{i=1}^{m} k_i \). Transition from one center to another is characterized by a routing probability \( r_{ij} \), i.e. the probability of going to center j on completion of service at center i. Jackson [JackJ63] has obtained
the equilibrium joint probability distribution of queue lengths for a broad class of queueing-theoretical models representing a network of service centers. Customer arrivals are modeled as a generalized Poisson process [cf. WagnH69], whose mean arrival rate varies almost arbitrarily with the total number of customers already in the system. Service completions at each center are also modeled as generalized Poisson processes, the mean service rate \( \mu \) at each center varying arbitrarily with the queue length there. Note that in Jackson's model all customers are identical. Muntz and Baskett [MuntR72] have a more general queueing network model that allows different classes of customers to have different branching probabilities. Gordon and Newell [GordW67] have presented a solution technique for closed queueing systems, i.e. networks of queues in which the number of customers is constant.

For closed queueing systems, Jackson's formulae for obtaining the equilibrium state probabilities are listed below.

\[
P(K) = \frac{w(K)}{T(S(K))}
\]

where,

\[
w(K) = \prod_{j=1}^{m} \prod_{i=1}^{k_j} \left[ \frac{e(i)}{u} \right] \text{ for } j \in [1,m]
\]

where \( e(j) = \sum_{i=1}^{m} e(i) \) for \( j \in [1,m] \)

\[
T(K) = \sum_{i \in S(K)} w(K) \text{ summed over } K \text{ with } S(K) = 1
\]

But, with \( P_c \) requests distributed uniformly and with the bus-bound situation or \( tp=tw \), the exponential server model reduces to \( m \) servers with
customers circulating with uniform routing probabilities i.e. \( r_{ij} = p_{ij} = 1/m \) Thus, 
\( e(j) \) denotes the average frequency of visits to service center \( j \). Using the above formulae we get,

\[
\begin{align*}
    w(K) &= (1/u)^n \\
    T(K) &= \left( \frac{n+m-1}{m-1} \right) (1/u)^n \\
    P(K) &= \left( \binom{n+m-1}{m-1} \right)^{-1}
\end{align*}
\]

for all \( K \) such that \( \sum_{i=1}^{m} k_i = n \)

i.e. all the states of the system have equal probability. Physically, this indicates that states with greater congestion in the queues are as likely as evenly distributed queues. Note that the above analysis holds even when successive memory requests are correlated as long as the average access frequency \( p_{ij} = 1/m \). The probability that a particular Mp module is idle, \( \text{Prob\{Mp[i] is idle\}} \), is the fraction of the total number of states that has \( k_1 = 0 \).

In other words,

\[
\text{Prob\{Mo[i] is idle\}} = \frac{\text{number of ways of assigning n Pa's to m-1 Mp's}}{\text{number of ways of assigning n Pa's to m Mp's}}
\]

Therefore,

\[
\text{Prob\{Mp[i] is busy\}} = \frac{n}{n+m-1}
\]

\[
\text{E[number of busy Mp's]} = m \cdot \text{Prob\{Mp[i] is busy\}} = \frac{mn}{m+n-1}
\]
2.1 Continuous Time Markov Chain Model

The above expression has a number of interesting properties: the expression is symmetric in \( m \) and \( n \); it has a basic hyperbolic form, asymptotic to \( n \) as \( m \) gets large; and, if we let \( m=n \) the above expression becomes

\[
\frac{n}{2-1/n}
\]

and

\[ E[\text{number of busy Mp's}] \to \frac{n}{2} \quad \text{for } n \gg 1 \]

The final observation has important implications. It states that as multiprocessor systems grow to include more and more Pc's, we are not faced with a law of diminishing returns: no matter how many Pc's are used, if we have the same number of memory modules, we can expect half the processors to be active.

2.2 A SIMPLE DISCRETE MARKOV CHAIN MODEL

For this analysis let us assume that all the Pc's are characterized by a single constant processing time \( t_p \). In this model, the memory access and cycle time are constant. The exponential server model discussed above allows the memory cycle time to have a large range of values. However, though the cycle time is not a constant (as seen by a Pc) it certainly does not have an exponential distribution. The constant service time model is an attempt to de-emphasize the small variance in the value of the cycle time. Although the processing time is not a constant in reality, this approach yields fairly good
estimates of the \( M_p/AR \) as substantiated in section 2.7. Also, all the memory units are assumed to have the same cycle time \( tc \) and access time \( ta \). Thus, the memory rewrite time is given by \( tw = tc - ta \). If \( tp = tw \) then all memory units can be considered to be operating synchronously. Thus, during any memory cycle the number of active \( P_c \)'s is equal to the number of busy \( M_p \)'s.

In this section, a simple Markov Chain Analysis is presented for the case in which the processors request every memory with equal likelihood. A multiprocessor system with \( n \) \( P_c \)'s and \( m \) \( M_p \)'s is likened to an occupancy problem with \( n \) balls and \( m \) urns. Balls are randomly assigned to the \( m \) urns at the beginning of a memory cycle. At the end of the cycle one ball is removed from each urn. Thus if there are \( k \) non-empty urns during cycle \( s \) then \( k \) balls are available for assignment during the \((s+1)\)-th cycle.

The state of the above mentioned process is defined by a \( m \)-tuple \((k_1, k_2, \ldots, k_m)\), where \( \sum_{i=1}^{m} k_i = n \) and \( 0 \leq k_i \leq n \) for all \( i \). The number of distinct states of the system is given by the combination \( \binom{n+m-1}{m-1} \) i.e. the number of ways in which \( n \) identical balls can be assigned to \( m \) bins [FellW66]. However, since all the processors behave identically, a number of the distinct states are equivalent i.e. they have the same occupancy and have the same components. e.g. states \((2,1,1)\), \((1,2,1)\), \((1,1,2)\) are equally likely. Thus, the reduced states are given by the different ways in which the number \( n \) can be partitioned into \( m \) parts. i.e. the unordered integer solutions to the equation \( \sum_{i=1}^{m} X_i = n \) for \( 0 \leq X_i \leq n \) represent equivalence classes of equally likely states. The number of such
TABLE 2.1

SOME PROPERTIES OF THE DISCRETE MARKOV CHAIN MODEL

<table>
<thead>
<tr>
<th>Number of Pc's</th>
<th>Number of Mp's</th>
<th>Total Number of States</th>
<th>Reduced States</th>
<th>Execution time for program</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td></td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>5</td>
<td></td>
<td>&lt; 1 sec.</td>
</tr>
<tr>
<td>8</td>
<td>6435</td>
<td>22</td>
<td></td>
<td>2 sec.</td>
</tr>
<tr>
<td>10</td>
<td>92378</td>
<td>42</td>
<td></td>
<td>8 sec.</td>
</tr>
<tr>
<td>12</td>
<td>1352878</td>
<td>77</td>
<td></td>
<td>1 min.</td>
</tr>
<tr>
<td>16</td>
<td>300540195</td>
<td>231</td>
<td></td>
<td>1 hour</td>
</tr>
</tbody>
</table>
partitions (for n≤m) is asymptotic to
\[ \frac{1}{4\pi^{3/2}} \exp\left[\frac{n(2n/3)^{1/2}}{2}\right] \quad \text{[cf. Beck64]} \]

Also,
\[ F(x) = \frac{1}{(1-x)(1-x^2) \cdots (1-x^k)} \]
\[ = 1 + \sum p(i)x^i \]

is an ordinary generating function of the sequence \((p(0), p(1), \ldots, p(k))\), where \(p(i)\) denotes the number of partitions of the integer \(i\) that have no parts exceeding \(k\), \(k≤i\) [Liu68]. Table 2.1 shows the total number of states and the number of reduced representative states as a function of \(n\).

Let the representative state \(S_i\) denote the set of compositions of the number \(n\) that yield the same partition e.g. the compositions \((2,1,1), (1,2,1)\) and \((1,1,2)\) correspond to the partition of the number 4 which has two 1's and one 2. Further, let \(S_{ij}\) be the individual compositions of the partition typified by representative state \(S_i\) and \(S_{i1}\) be that composition which has its components arranged in monotone non-decreasing order, i.e. \((2,1,1)\) for the above example. The algorithm shown in Fig 2.1 generates all the partitions of \(n\) with the components in monotone non-increasing order.

Let \(X_{ij}\) denote the probability of a transition from \(S_j\) to \(S_i\). Then, due to the symmetry of the problem,
START
Number to be partitioned...N
Maximum number of parts...M

\[ K \leftarrow 1; A(1) \leftarrow N \]

\[ A(I) \leftarrow 1 \text{ for } I=2, \ldots, K \]

\[ A(1) \leftarrow N - \sum_{I=2}^{K} A(I) \]

Components of a partition are given by \( A(I), I=1, \ldots, K \). This partition has \( K \) non-zero parts. Store or display this partition.

\[ T \leftarrow 2 \]

\[ T=K+1? \]

\[ A(I) - A(T) > 1? \]

\[ T \leftarrow T+1 \]

\[ A(I) \leftarrow A(T) + 1 \text{ for } I=2, \ldots, T-1 \]

\[ K \leftarrow K + 1 \]

\[ K \leq \text{Min}(M,N)? \]

STOP

Figure 2.1 An algorithm for generating partitions
Chapter 2: Multiprocessors with \( tp=tw \)

2.2 Discrete Markov Chain Model

Let the \( m \)-tuple \((k_1, k_2, \ldots, k_m)\) denote the state of the Markov chain. If \( x \) is the number of non-zero elements in this vector then at the end of the memory cycle, \( x \) new processors have to be reassigned to memory modules. At the end of the current memory cycle the queue is characterized by the partial state \( m \)-tuple \((j_1, j_2, \ldots, j_m)\), where

\[
j_1 = k_1 - 1 \text{ if } k_1 > 0
\]

\[
= 0 \text{ otherwise.}
\]

A new state \((l_1, l_2, \ldots, l_m)\) is reachable from \((k_1, k_2, \ldots, k_m)\) if and only if \( l_i \geq j_i \) for \( 1 \leq i \leq m \). If the above condition is satisfied the probability of the state transition is given by

\[
X_{ij} = \sum \text{Prob}\{\text{Transition from } S_i.l \text{ to } S_i.k\}
\]

\[
S_i.k \quad S_i
\]

\[
\sum_{S_i.k} X_{ij} = \sum \text{Prob}\{\text{Transition from } S_i.l \text{ to } S_i.k\}
\]

\[
S_i.k \quad S_i
\]

Thus, we now have a formula for generating the transition probabilities. Due to the symmetry of the problem it suffices to generate only the transition
Figure 2.2 Next states accessible from initial state (2,2,0,0)
probabilities for the representative class of states. All the different ways of obtaining the same partition are lumped together to form a reduced state.

To illustrate a computational method** for generating the transition probabilities consider an example of a 4 by 4 system. The number 4 can be partitioned in 5 different ways as listed below:

- 4 0 0 0
- 3 1 0 0
- 2 2 0 0
- 2 1 1 0
- 1 1 1 1

These partitions represent 5 equivalence classes that characterize the state of the Markov Chain. Let us consider the state (2,2,0,0). At the end of a memory cycle, the resultant partial state is (1,1,0,0) with 2 free processors to be reassigned. Figure 2.2 shows the different ways in which these 2 Pcs can be assigned, one at a time, to reach a new partial representative state. After both Pcs are assigned a terminal state is reached. The number on the arrow indicates the number of ways of reaching the partial or terminal state that the arrow points to. Now the number of ways in which a final state can be reached from the

**The use of a tree to generate the transition probabilities was suggested by F. Baskett and D. Chewning of Stanford University.
Figure 2.3 Enumeration tree for a 4 by 4 multiprocessor system.
initial state can be computed by traversing the tree, e.g. there are $2 \times 1$ ways of reaching $(1,1,1,1)$ and $(2 \times 2 + 2 \times 3)$ ways of reaching $(2,1,1,0)$ from $(2,2,0,0)$.

It is possible to construct a single tree with different pointers for different initial states. Figure 2.3 shows a complete tree for a 4x4 system. Initial states are circled. The entire transition matrix can be filled by traversing this tree. A convenient way of traversing this tree is by using a stack which has depth equal to one more than the number of PC's. At each level the stack contains a partial state and has a pointer to the initial representative state (if any) from which it is derived. The stack is initialized to contain the path that leads to the topmost final state. For this example the stack is initialized as shown in Fig. 2.4, and Fig. 2.5 shows an algorithm† for using the tree to generate the transition matrix, shown in Fig. 2.6.

The tree in Fig. 2.3 can be converted into a mesh by lumping together all occurrences of a partial state in the tree, e.g. state 2100 at level 3 appears twice. The resulting mesh for the 4 by 4 example is shown in Fig. 2.7. The algorithm for generating the transition matrix is shown in Fig. 2.8. Though the implementation of this algorithm‡‡ involves a matrix multiplication and requires

†A FORTRAN implementation of this algorithm is listed in Appendix A-1.

‡‡See Appendix A-2 for a listing of a FORTRAN implementation.
Figure 2.4 Initial contents of the stack for traversing the tree shown in figure 2.3

<table>
<thead>
<tr>
<th>Initial state pointer</th>
<th>STACK</th>
<th>NWAYS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 0 0 0</td>
<td>4 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>3 1 0 0</td>
<td>3 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>2 1 1 0</td>
<td>2 0 0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>1 0 0 0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0 0 0 0</td>
<td>X</td>
</tr>
</tbody>
</table>

Number of ways of getting to level L from level L-1
Transition matrix has been completely generated.

Figure 2.5 Algorithm for traversing the tree shown in Figure 2.3
STEP 1: X_{ij} is the number of ways of reaching i from j.
(Obtained from the tree of fig. 2.3) by using the

STEP 2: \[ X_{ij} = \frac{X_{ij}}{\sum_{i} X_{ij}} \quad \text{(Note that } \frac{1}{\sum_{i} X_{ij}} \text{, where } x \text{ of the } \sum_{i} X_{ij} \text{ components of } j \text{ are non-zero)}\]

Final equations to be solved simultaneously:

\[
\begin{bmatrix}
P_{4000} \\
P_{3100} \\
P_{2200} \\
P_{2100} \\
P_{1111}
\end{bmatrix} = 
\begin{bmatrix}
0.25 & 0.0625 & 0.000 & 0.015625 & 0.015265 \\
0.75 & 0.3750 & 0.125 & 0.187500 & 0.187500 \\
0.00 & 0.1875 & 0.125 & 0.140625 & 0.140625 \\
0.00 & 0.3750 & 0.625 & 0.562500 & 0.562500 \\
0.00 & 0.0000 & 0.125 & 0.093750 & 0.093750
\end{bmatrix}
\begin{bmatrix}
P_{4000} \\
P_{3100} \\
P_{2200} \\
P_{2100} \\
P_{1111}
\end{bmatrix}
\]

Subject to: \[ P_{4000} + P_{3100} + P_{2200} + P_{2100} + P_{1111} = 1 \]

Figure 2.6 Steps in the generation of the transition matrix
more temporary storage it is faster than the algorithm in section 2 for large n. Thus, a space-time trade-off affects the selection of the algorithm to be used.

The following theorem and lemma can be used to increase the efficiency of the program that generates the transition probabilities.

**Theorem 1:** There is a one-to-one correspondence between a representative state and a partial state that the representative state reduces to at the end of a cycle.

**Proof:** Let \((k_1, k_2, ..., k_m)\) be a representative state. The partial state at the end of the cycle is given by

\[
(j_1, j_2, ..., j_m)
\]

where

- \(j_1 = k_1 - 1\) if \(k_1 > 0\)
- \(= 0\) otherwise

Since no two representative states are alike and \(\sum_{i=1}^{m} k_i = n\), it follows that the partial states are distinct.

**Lemma:** A partial state at level \(L\) in the enumerative tree of Fig. 2.3 can correspond to a terminal state with exactly \(n-L\) occupied \(M_p\)'s.
Fig. 2.7. Enumeration mesh for a 4 by 4 multiprocessor system.
generate partitions of N into M parts, 
let the vector XSYS denote these partitions, 
that represent the state of the system.

Generate partitions of N-1 into M parts. 
Let X2 denote these partial states. 
\[ I = N-1 \]

Compute B. 
Matrix B is the number of ways of reaching XSYS from X2.

Update TRANS (i.e. matrix of number of ways of reaching XSYS from XSYS.) 
if any state in X2 is a reduction of a state in XSYS.

I=I-1

I=0?  \[ Y \]

Generate partitions of I 
Vector X1

Compute matrix A. 
A is the number of ways of reaching X2 from X1.

Matrix Multiplication. 
B = AxB 
Interchange X1 and X2. 
B is now the number of ways of reaching XSYS from X2.

If 00...0 is a reduction of a state in XSYS update TRANS.

Evaluation of the required matrix of the number of transitions, TRANS is complete.

Fig. 2.8 An algorithm for evaluating the transition matrix.
Chapter 2 : Multiprocessors with $tp=tw$

2.3 Discrete Markov Chain Model

Proof: Let $J=(j_1, j_2, ..., j_m)$ be a partial state in the tree depicted in Fig. 2.3. Furthermore, let the number of non-zero elements in the partial state be $y$ and let $\sum_{i=1}^{m} j_i = n - x$. Since one Pc is always removed from a non-empty queue at the end of a cycle, $J$ is a partial state that can be reduced from a valid representative state $K=(k_1, k_2, ..., k_n)$, if and only if

(i) The number of non-zero elements in $K$ is $x$, and

(ii) $x > y$

Note that $x$ and $y$ are both less than or equal to $\min(m, n)$ and $\sum_{i=1}^{m} k_i = n$. Then, if $x > y$, $J$ has at least $x - y$ zeros. If $x \leq y$, then there is no representative state $K$ that corresponds to the partial state $J$. If $x > y$, then the representative state is obtained by adding $y$ 1's to the non-zero elements of $J$ and replacing $x - y$ zeros of $J$ by 1. At level $L$, $\sum_{i=1}^{m} j_i = L$. Therefore, $x$, the number of occupied Mp's in $K$, is equal to $n - L$.

Figure 2.9 shows the average number of busy Mp's when $n = m$. The curve has an almost constant slope of .586 for $n > 4$. Thus, this model also shows the absence of a law of diminishing returns. Figures 2.10 and 2.11 show the effect of adding a Pc and an Mp respectively on the average number of busy Mp's. Also, the average number of busy Mp's is almost symmetrical with respect to $m$ and $n$. The results obtained from the model are compared with a less restrictive simulation model in section 2.7.
Figure 2.10 The effect of adding a PC.
2.3 APPROXIMATE DISCRETE MARKOV CHAIN MODELS

Even with the representative state approach the number of states characterizing the Markov Chain increases rapidly as \( n \) increases. Table 2.1 shows the number of representative states as a function of \( n \) and the approximate execution time needed on a DEC PDP-10 for the FORTRAN program listed in Appendix A-2, which determines the stationary state probabilities. Though the analysis is exact, the size of the problem (as indicated by the array space used by the program) and the time required restricts the use of the model described in section 2.2.

2.3.1 A New Approximate Discrete Markov Chain Model

Because of the high cost of computation for the previous model, an approximate discrete Markov chain model will now be proposed, and the results of section 2.2 will be used to improve the applicability of this new approximate model. The state is denoted by the number of active Pc's. Thus the number of states is \( \min(n,m) \). Note that only those Pc's that are active during the current cycle make new requests during the next cycle. Also, the number of busy memories is equal to the number of active processors. The approximation propounded here consists of removing the non-active Pc's from the Mp queues and reassigning them as indicated below. This approach was motivated by a gross intuitive feeling.
that if the Pc's are removed from the queues and asked to make new requests, they would end up in the same queues as before. However, this is not exactly true, and the heavily congested states tend to be de-emphasized. Let the number of busy Mp's during the current cycle be \( i \). Then, during the next cycle the \( i \) active Pc's make a new request to the \( m \) Mp's and some of the \( n-i \) non-active Pc's get serviced if they are at the front of the queue. However, in this approximate model, the \( n-i \) non-active Pc's are removed from the \( i \) Mp queues and reassigned to the same \( i \) queues. This is equivalent to the \( n-i \) Pc's making new requests to the \( i \) Mp's. Thus the \( i \) Pc's may not end up in the same queues that they were removed from. This approximation will be used widely in this thesis. The results of this section show the accuracy of the approximation.

Now, the probability that \( j \) out of the \( i \) active Pc's make a new request at the beginning of the next cycle to one of the \( i \) Mp's that are busy during the current cycle, is given by

\[
X_{\text{PROB}} = \binom{i}{j} \cdot \left( \frac{j}{m} \right) \cdot \left( 1-\frac{j}{i} \right)^{i-j}
\]

Thus, during the next cycle, with probability \( X_{\text{PROB}} \) the \( n-i \) non-active Pc's and \( j \) active Pc's are assigned to the \( i \) busy Mp's of the current cycle, and the remaining \( i-j \) active Pc's make a request to the other \( m-i \) Mp's.

Let \( n_n = n-i+j \) and \( k_n = \min(n,ij) \). Note that \( n_n \) denotes the number of Pc's that will be queued (during the next cycle) for the \( i \) Mp's that are busy during the current cycle. Also, let \( X(I) \) denote the conditional probability that \( I \) out of \( i \) busy Mp's are also busy during the next cycle, given that \( n-i+j \) Pc's will be
Chapter 2: Multiprocessors with \( t_p = t_w \)

2.3 Approximate Discrete Markov Chain Models

queued for the \( i \) Mp's. The number of ways that \( n \) different Pc's can be assigned to \( i \) different Mp queues is \( i^n \) [Rior.58, pp.90]. Also the number ways that the \( n \) Pc's can access \( i \) Mp's so that exactly \( l \) Mp's are occupied and \( i-1 \) are not is given by Riordan [Rior.58] as:

\[
CM(i,l) * S(nn,l)
\]

where \( CM(i,l) = i(i-1) \ldots (i-l+1) \)

and \( S(nn,l) \) is the Stirling\( \dagger \) number of the second kind

Thus,

\[
X(l) = CM(i,l) * S(nn,l)/i^n
\]

Now, let \( Y(l_2) \) be the conditional probability that \( l_2 \) out of the \( m-i \) currently non-busy Mp's are busy during the next cycle, given that \( i-j \) Pc's make a request. Then,

\[
Y(l_2) = CM(m-i,l_2) * S(i-j,l_2)/\binom{m-i}{l_2}
\]

Thus, the probability that \( k = l_1 + l_2 \) Mp's will be busy during the next cycle is

\[X \text{PROB} * X(l_1) * Y(l_2)\]

Therefore, \( \text{TRANS}(k,i) \), the probability of a transition from current state \( i \) to next state \( k \)

\[\dagger \text{Stirling Numbers of the second kind are used to convert from powers to binomial coefficients.}\]

\[
x^n = \sum_k S(n,k) \binom{k}{j} k!
\]

Also,

\[
S(i,j) = j * S(i-1,j) + S(i-1,j-1)
\]

with \( S(i,0) = S(0,j) = 0 \)

and \( S(i,i) = 1 \)
TABLE 2.2

Comparison of Exact and Approximate Models

Approximate Discrete Markov Chain Model for $tp=tw$

Average Number of Busy Mps

<table>
<thead>
<tr>
<th></th>
<th>$m=2$</th>
<th>$m=4$</th>
<th>$m=8$</th>
<th>$m=16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=2$</td>
<td>1.5000</td>
<td>1.7500</td>
<td>1.8750</td>
<td>1.9735</td>
</tr>
<tr>
<td>$n=4$</td>
<td>1.8000</td>
<td>2.6550</td>
<td>3.2751</td>
<td>3.6291</td>
</tr>
<tr>
<td>$n=8$</td>
<td>1.9846</td>
<td>3.4858</td>
<td>5.0999</td>
<td>6.3680</td>
</tr>
<tr>
<td>$n=16$</td>
<td>1.9999</td>
<td>3.9343</td>
<td>6.8436</td>
<td>10.0058</td>
</tr>
</tbody>
</table>

Exact Discrete Markov Chain Model

Average Number of Busy Mps

<table>
<thead>
<tr>
<th></th>
<th>$m=2$</th>
<th>$m=4$</th>
<th>$m=8$</th>
<th>$m=16$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=2$</td>
<td>1.5000</td>
<td>1.7500</td>
<td>1.8750</td>
<td>1.9735</td>
</tr>
<tr>
<td>$n=4$</td>
<td>1.7500</td>
<td>2.6210</td>
<td>3.2652</td>
<td>3.6268</td>
</tr>
<tr>
<td>$n=8$</td>
<td>1.8750</td>
<td>3.2657</td>
<td>4.9471</td>
<td>6.3149</td>
</tr>
<tr>
<td>$n=16$</td>
<td>1.9375</td>
<td>3.6270</td>
<td>6.3154</td>
<td>9.6258</td>
</tr>
</tbody>
</table>
2.3 Approximate Discrete Markov Chain Models

\[ \sum \text{PROB} \times \text{X(i)Y(i)} \]

the summation is over the different ways of choosing
1, and l2, such that k = l1 + l2.

A FORTRAN program that determines the steady state probabilities is listed
in Appendix A-3. Table 2.2 shows the average number of busy Mp's as predicted by
this approximate model. Due to the small number of states, this approximate
model needs much less computer time; typically about 1 second of execution time
on a PDP-10 for a 16x16 multiprocessor system. Table 2.2 shows that the average
number of busy Mp's is almost symmetric in m and n. The approximate model has a
larger error for n>m. Therefore, a better estimate of the performance of a nxm
system can be obtained by evaluating the performance of a mxn system if n>m, a
conclusion possible only due to the results of the exact analysis of section
2.2.

2.3.2 Strecker’s Approximation

Strecker [StreW70] has an approximate closed form solution to the discrete
Markov Chain model presented here. His approach is equivalent to removing the
queued processors from all the memory modules at the end of a memory cycle and
reassigning them among all the memory modules. In the approximate model proposed
earlier in section 2.3.1, the Pc's that were queued at the end of the cycle were
reassigned only among the busy Mp's. Thus, Strecker’s analysis is more
TABLE 2.3

Expected number of busy memories in one cycle
Number of Pc's = 1,2,...,8 (rows)
Number of Mp's = 1,2,...,8 (columns)

Discrete Markov Chain Model

<table>
<thead>
<tr>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1.5000</td>
<td>1.6667</td>
<td>1.7500</td>
<td>1.8000</td>
<td>1.8333</td>
<td>1.8571</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.6667</td>
<td>2.0476</td>
<td>2.2692</td>
<td>2.4095</td>
<td>2.5054</td>
<td>2.5748</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.7500</td>
<td>2.2781</td>
<td>2.6218</td>
<td>2.8630</td>
<td>3.0365</td>
<td>3.1657</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.8333</td>
<td>2.5859</td>
<td>3.0370</td>
<td>3.4533</td>
<td>3.7809</td>
<td>4.0415</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.8571</td>
<td>2.5751</td>
<td>3.1663</td>
<td>3.6486</td>
<td>4.0418</td>
<td>4.3636</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.8750</td>
<td>2.6274</td>
<td>3.2657</td>
<td>3.8024</td>
<td>4.2521</td>
<td>4.6294</td>
</tr>
</tbody>
</table>

Strecker's Approximation

<table>
<thead>
<tr>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
<th>1.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000</td>
<td>1.5000</td>
<td>1.6667</td>
<td>1.7500</td>
<td>1.8000</td>
<td>1.8333</td>
<td>1.8571</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.7500</td>
<td>2.1111</td>
<td>2.3125</td>
<td>2.4400</td>
<td>2.5278</td>
<td>2.5918</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.8750</td>
<td>2.4074</td>
<td>2.7344</td>
<td>2.9520</td>
<td>3.1065</td>
<td>3.2216</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.9375</td>
<td>2.6049</td>
<td>3.0588</td>
<td>3.3616</td>
<td>3.5887</td>
<td>3.7613</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.9687</td>
<td>2.7366</td>
<td>3.2881</td>
<td>3.6893</td>
<td>3.9966</td>
<td>4.2240</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.9844</td>
<td>2.8244</td>
<td>3.4661</td>
<td>3.9514</td>
<td>4.3255</td>
<td>4.6206</td>
</tr>
<tr>
<td>1.0000</td>
<td>1.9922</td>
<td>2.8829</td>
<td>3.5995</td>
<td>4.1611</td>
<td>4.6046</td>
<td>4.9605</td>
</tr>
</tbody>
</table>

Percentage Error

<table>
<thead>
<tr>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
<th>0.0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>0.0000</td>
<td>4.9979</td>
<td>3.1012</td>
<td>1.9822</td>
<td>1.2658</td>
<td>0.8941</td>
<td>0.6602</td>
<td>0.5128</td>
</tr>
<tr>
<td>0.0000</td>
<td>7.1429</td>
<td>6.0482</td>
<td>4.3266</td>
<td>3.1086</td>
<td>2.3053</td>
<td>1.7658</td>
<td>1.3874</td>
</tr>
<tr>
<td>0.0000</td>
<td>7.6389</td>
<td>8.0782</td>
<td>6.5484</td>
<td>5.0631</td>
<td>3.9299</td>
<td>3.1082</td>
<td>2.4935</td>
</tr>
<tr>
<td>0.0000</td>
<td>7.3856</td>
<td>9.2863</td>
<td>8.2680</td>
<td>6.8340</td>
<td>5.5463</td>
<td>4.5157</td>
<td>3.7114</td>
</tr>
<tr>
<td>0.0000</td>
<td>6.8548</td>
<td>9.6812</td>
<td>9.4685</td>
<td>8.2991</td>
<td>7.0191</td>
<td>5.8896</td>
<td>4.9512</td>
</tr>
<tr>
<td>0.0000</td>
<td>6.2507</td>
<td>9.7244</td>
<td>10.2214</td>
<td>9.4335</td>
<td>8.2900</td>
<td>7.1521</td>
<td>6.1450</td>
</tr>
</tbody>
</table>
approximate and will underestimate the interference. However, Strecker obtains his approximate solution in a closed form, which will be modified here to yield more accurate estimates of the \( \frac{M_p}{AR} \). Thus the state of the system is considered independent of the state during the last cycle. If we use this assumption the distribution of \( \text{Pc}'s \) queued for an \( M_p \) follows the binomial distribution:

\[
\text{Prob}(Y = r) = \binom{n}{r} \left( \frac{1}{m} \right)^r \left( 1 - \frac{1}{m} \right)^{n-r}
\]

where \( Y \) is a random variable equal to the number of \( \text{Pc}'s \) queued for \( M_p[j] \) and \( p_{ij} = 1/m \) for all \( i \) and \( j \).

Thus,

\[
\text{Prob}(M_p[j] \text{ is busy}) = 1 - \text{Prob}(\text{nobody is queued for } M_p[j])
\]

\[
= 1 - (1 - \frac{1}{m})^n
\]

In other words, the occupancy of \( M_p[j] \) is \( 1 - (1 - \frac{1}{m})^n \), and

\[
E[\text{no. of occupied } M_p's] = \sum_{j=1}^{m} \text{[Occupancy of } M_p[j])
\]

\[
= m \times [1 - (1 - \frac{1}{m})^n]
\]

Table 2.3 shows a comparison of Strecker's results and the exact Markov chain analysis. Note that Strecker's results are optimistic estimates of the unit execution rate. It is encouraging to note that such a simple expression is within 6 to 8% of the exact Markov chain model for \( m/n > 0.75 \). This is because his analysis assumes that all \( n \) \( \text{Pc}'s \) always make a new request at the beginning of
Figure 2.12 Streckeck’s formula for fixed \( m \)

\[ f(n) = n \cdot e^{n \cdot \ln\left(1 - \frac{1}{m}\right)} \]
Chapter 2: Multiprocessors with \( tp=tw \)

2.3 Approximate Discrete Markov Chain Models

Each memory cycle, whereas in the discrete Markov chain only those \( Pc \)'s that receive service are allowed to make new requests. Moreover, note that the expression \( m*[1-(1-1/m)^n] \) can be written in an exponential form as

\[
m*[1-\exp[n* \ln (1-1/m)]]
\]

Figure 2.12 shows a plot of the above expression for fixed \( m \); the relaxation time \( [\ln (1-1/m)]^{-1} \) approaches \( m \) as \( m \) gets large.

The exact discrete Markov chain model of section 2.2 shows the performance to be almost symmetric in \( n \) and \( m \). Also, the analysis of the error of Strecker's approximation, shown in Table 2.4, indicates a greater accuracy for \( n < m \). Thus, a more accurate estimate of the average number of busy \( Mp \)'s is \( \frac{1}{i} \left[ 1-\frac{1}{i-1} \right] \), where \( i=\max(n,m) \) and \( j=\min(m,n) \). Note that the above formula was not derived by Strecker. It was possible to obtain it due to the knowledge gained from the exact analysis presented in section 2.2.

2.4 DISCRETE MARKOV CHAIN MODEL OF SKINNER AND ASHER

Skinner and Asher [SkinC69] model the multiprocessor system with \( tp=tw \) as a discrete Markov chain. They assume a matrix of probabilities that express the likelihood that a given processor requests service from a given memory at the beginning of a memory cycle, provided the \( Pc \) is not queued. They also assume a matrix of probabilities that express the likelihood of the various outcomes that
can arise when there are simultaneous requests to one memory by several processors. The state of the system is characterized by the processors queued for the different memory modules. A state transition matrix is formed from the access probabilities and the steady state probabilities of various states are determined by solving the state transition equations. The number of states of the system increases very steeply with an increase in the number of Pcs and Mps. Closed form solutions are presented only for cases with up to 2 Pcs and n Mps. The analysis in the previous section is similar to Skinner and Asher, but with uniformly random access patterns for all the Pcs, i.e. \( p_{ij} = 1/m \) for all \( i \). The results of Skinner and Asher are compared with a new approximate model in section 2.6.

2.5 DIFFUSION APPROXIMATIONS

An approximation method that has been proposed for the solution of general queueing networks is the diffusion approximation [cf. NeweG71; KobaH73]. A discrete-state process is approximated by a Wiener-Levy diffusion process with a continuous path. The key assumption in such an analysis is that incremental changes in the queue lengths are normally distributed. This leads to a characterization of the queueing network by a set of diffusion equations. The accuracy of the approximation depends on three factors: (i) approximation of a
discrete-state process by a time-continuous Markov process, (ii) choice of proper reflecting barriers, and (iii) discretization of the continuous density function for queue lengths. Surprisingly, for the simple discrete Markov Chain model of section 4, the diffusion approximation yields a result identical to that with exponential servers derived from Jackson's formulae. However, the main utility of the diffusion approximation in this context is that it can be used to analyze the effect of different coefficients of variation (ratio of standard deviation to the mean) for the service time distribution. Unfortunately, for $P_{ij}={1/m}$, the diffusion approximation predicts that the average number of busy memories to be independent of the service time distribution as long as all servers are identical. Thus, the diffusion approximation has proved to be a disappointing tool in this study.

2.6 AN APPROXIMATE MODEL FOR ARBITRARY $P_{ij}$

In this section, we explore the effect of non-uniform access probabilities (i.e. $P_{ij}$ is no longer restricted to be equal to $1/m$) on the $M_{pAR}$. This situation often arises in physical systems in which each $P_c$ has a greater preference for a different memory module. Now, we analyze multiprocessor systems in which $F_{ij}$ can take any arbitrary value between 0 and 1 subject to $\sum_{j=1}^{m} P_{ij}=1$. Let $Q_{ij}$ denote the probability that processor $i$ is queued for memory $j$. The probability that memory $j$ is busy or occupied is given by
Chapter 2: Multiprocessors with \( t_p = t_m \)

2.6 Approximate Model for Arbitrary \( P_{ij} \)

1 - \( \text{Prob(} \text{no processor is queued for memory } j \text{)} \)

\[
1 - \prod_{i=1}^{n} (1 - Q_{ij})
\]

assuming that the event of a \( P_i \) not being queued for an \( M_j \) is independent of other \( P_i \)'s not being queued. The simulation results shown later justify this assumption.

The above denotes the average number of requests serviced in one memory cycle.

Thus, if, \( M \) is the expected value of the number of occupied memories during a memory cycle,

\[
M = \sum_{j=1}^{m} \text{Prob} \left( M_p[j] \text{ is occupied} \right)
\]

\[
= \sum_{j=1}^{m} \sum_{i=1}^{n} \left[ 1 - \prod_{i=1}^{n} (1 - Q_{ij}) \right]
\]

In general, the probabilities \( Q_{ij} \) and \( P_{ij} \) are not equal. The \( P_{ij} \)'s are a characteristic of each processor and therefore independent of the behavior of the other processors in the system. However, any \( Q_{ij} \) is a function of all the \( P_{ij} \)'s of the multiprocessor system. Strecker (StreW70) has evaluated the unit execution rate of a multiprocessor system in which \( P_{ij} = 1/m \) for all values of \( i \) and \( j \). He makes no attempt to obtain a relationship between \( Q_{ij} \) and \( P_{ij} \). Strecker assumes that \( P_{ij} = Q_{ij} \) and states that his results are approximate. The approximation in his analysis is due to the assumed binomial distribution for
the queued processors. If all the Pcs are identical and have equal likelihood of accessing every memory unit, then the probability of any processor being queued for any memory is uniformly equal. Since the memories operate synchronously and all requests occur at the end of a memory cycle a processor is always queued. Hence, the probability $Q_{ij} = 1/m$.

Let us focus our attention on $Pc[i]$ and $Mp[j]$. The time spent by $Pc[i]$ in queue for $Mp[i]$ depends on $P_{ij}$ and $Q_{ij}$, $i \neq i$. Thus, $Q_{ij}$ depends on other $Q_{ik}$'s, which in turn depend on $Q_{ij}$. Let us for a moment allow other processors to make requests to memory before $Pc[i]$, and let $Y_{ij}$ denote the probability that none of the other n-1 Pcs' request service from $Mp[j]$. Thus,

$$Y_{ij} = \prod_{k \neq i} (1 - P_{kj})$$

Now, if none of the other Pcs make a request to $Mp[j]$ the waiting time (including service) is one cycle time. However, if other Pcs make a request to $Mp[j]$ before $Pc[i]$, then $Pc[i]$ has to wait for those Pcs to be served. Now, let us look at $Pc[l]$, which has to wait for service from $Mp[j]$ if other processors make a request before it does. Here, we shall allow other Pcs to make requests before $Pc[l]$. Thus, $Pc[l]$ waits in queue for $Mp[j]$ with probability $P_{lj} * (1 - Y_{lj})$. Thus, the average number of Pcs that $Pc[i]$ finds waiting before itself is $\sum P_{ij} * (1 - Y_{ij})$. However, $Pc[i]$ accesses $Mp[j]$ with probability $P_{ij}$. Therefore, the weighted waiting time $T_{ij}$ for $Pc[i]$ in queue for $Mp[j]$ is

$$P_{ij} * (Y_{ij} + [1 + \sum_{l \neq i} P_{lj} * (1 - Y_{lj})] * (1 - Y_{ij}))$$

Therefore, the average time is equal to $\sum_{j=1}^{m} T_{ij}$ and
Figure 2.13 Comparison of approximate model and Skinner and Asher's results.
2.6 Approximate Model for Arbitrary $P_{ij}$

\[ Q_{ij} = \frac{T_{ij}}{T} \]

Figure 2.13 compares the execution rate predicted by the approximate model of this section with the exact Markov chain model of Skinner and Asher for a 2x2 multiprocessor system. $P_{ij}$ is equal to $\alpha$ for $i=j$ and equal to $\beta$ for $i \neq j$.

Now, suppose each Pc has a greater preference for one Mp. Let us use the model to examine the effect of assigning access probabilities so that $P_{ij}=\alpha(1/m)$ for $i=j$ and $P_{ij}=\beta=(1-\alpha)/(m-1)$ for $i \neq j$. Note that $\beta<1/m$. Thus, each Pc has a greater preference for a certain memory module. For example, the access probability matrix for a 4x5 multiprocessor system is shown below.

\[
\begin{bmatrix}
\alpha & \beta & \beta & \beta & \beta \\
\beta & \alpha & \beta & \beta & \\
\beta & \beta & \alpha & \beta & \\
\beta & \beta & \beta & \alpha & \beta
\end{bmatrix}
\]

Figures 2.14 and 2.15 show the effect of changing $\alpha$ from 0 to 1 for a 8x16 and a 16x16 multiprocessor system. Also, Table 2.4 compares the results predicted by the model with simulation results, because the analysis is approximate. Note that both graphs show that the execution rate is a minimum for $\alpha=1/m$. With $\alpha=1/m$ this model predicts the same result as Strecker's approximation as both models assume a binomial distribution for the queued processors. The approximation error reduces as $\alpha$ increases, the error being zero for $\alpha=1$. An error correction factor can be used as described below. For $\alpha=1/m$
Figure 2.14 The Effect of $\alpha$ on the execution rate of a 8x16 Multiprocessor System.
Figure 2.15 The Effect of $\gamma$ on the execution rate of a 16x16 multiprocessor system.
TABLE 2.4

Comparison of simulation results with analytic model of Section 2.3

A 16x16 Multiprocessor System

<table>
<thead>
<tr>
<th>α</th>
<th>90% confidence interval from simulation</th>
<th>Analytic Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>(9.7482, 9.8329)</td>
<td>10.4174</td>
</tr>
<tr>
<td>0.50</td>
<td>(10.5824, 10.5946)</td>
<td>10.9973</td>
</tr>
<tr>
<td>0.75</td>
<td>(12.0875, 12.3907)</td>
<td>12.5323</td>
</tr>
<tr>
<td>0.90</td>
<td>(13.9111, 14.2145)</td>
<td>14.3873</td>
</tr>
</tbody>
</table>

A 8x16 Multiprocessor System

<table>
<thead>
<tr>
<th>α</th>
<th>90% confidence interval from simulation</th>
<th>Analytic Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>(6.2803, 6.4139)</td>
<td>6.4880</td>
</tr>
<tr>
<td>0.50</td>
<td>(6.5374, 6.5614)</td>
<td>6.6885</td>
</tr>
<tr>
<td>0.75</td>
<td>(6.9693, 7.0853)</td>
<td>7.1673</td>
</tr>
<tr>
<td>0.90</td>
<td>(7.4338, 7.6041)</td>
<td>7.6328</td>
</tr>
</tbody>
</table>

A 8x8 Multiprocessor System

<table>
<thead>
<tr>
<th>α</th>
<th>90% confidence interval from simulation</th>
<th>Analytic Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>(4.8886, 5.1126)</td>
<td>5.2798</td>
</tr>
<tr>
<td>0.50</td>
<td>(5.2797, 5.4127)</td>
<td>5.5372</td>
</tr>
<tr>
<td>0.75</td>
<td>(6.0926, 6.1901)</td>
<td>6.2807</td>
</tr>
<tr>
<td>0.90</td>
<td>(6.9245, 7.1411)</td>
<td>7.1975</td>
</tr>
</tbody>
</table>
the exact Markov chain model should be used to compute the execution rate $X_i$; the approximate model of this section predicts an execution rate, $X_j$, equal to $m(1-(1-1/m)^n)$ for $\alpha = 1/m$. The correction factor for $\alpha = 1/m$ is $X_j/X_i$. Let $e = (X_j - X_i)/X_j$. Then a linear error correction factor $F$ is $1-e(1-\alpha)/(1-1/m)$ for $\alpha > 1/m$. The corrected estimate is $F \times X_j$, where $X$ is the execution rate for the given value of $\alpha$. The dotted lines in figures 2.14 and 2.15 show the corrected execution rates. The vertical lines show 90% confidence intervals obtained by simulation. This model shows the increase in the $M_pAR$ due to deskewing of the processors’ access patterns.

2.7 CONCLUDING REMARKS

Tables 2.3 and 2.5 compare the numerical results obtained from the different models described. Note that the continuous and discrete Markov chain models exhibit similar trends, though the numerical values differ. Strecker’s approximation gets better as $m/n$ increases, whereas the continuous time and discrete Markov models get closer for larger $n/m$ ratios. Table 2.6 shows some simulation results obtained with exponential distributions for the processing.

\[\text{\textsuperscript{+All confidence intervals in this thesis will be shown by vertical lines. Unless specified otherwise, the confidence intervals are calculated from about 10 independent samples, each averaged over about 3000 cycles.}\]
### TABLE 2.5

Expected number of busy memories in one cycle

Number of PCs = 1, 2, ..., 8 (rows)
Number of MCs = 1, 2, ..., 8 (columns)

<table>
<thead>
<tr>
<th></th>
<th>Discrete Markov Chain Model</th>
<th>Continuous Time Markov Chain Model</th>
<th>Percentage Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>1.0000</td>
<td>1.0000</td>
<td>0.0000 0.0000 0.0000</td>
</tr>
<tr>
<td>1.0000 1.5000</td>
<td>1.0000 1.6667</td>
<td>1.0000</td>
<td>1.0000 0.0888 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>1.6667 1.7500</td>
<td>1.0000</td>
<td>1.0000 1.0000 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>2.0476 2.2692</td>
<td>1.0000</td>
<td>1.0000 1.5888 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>2.2701 2.6210</td>
<td>1.0000</td>
<td>1.0000 1.8880 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>2.4102 2.8633</td>
<td>1.0000</td>
<td>1.0000 1.8880 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>2.5859 3.0370</td>
<td>1.0000</td>
<td>1.0000 1.8880 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>2.5674 3.2657</td>
<td>1.0000</td>
<td>1.0000 1.8880 0.0000</td>
</tr>
<tr>
<td>1.0000 1.0000</td>
<td>1.8888 1.8888 1.8880 1.8888</td>
<td>1.0000</td>
<td>1.0000 1.8880 0.0000</td>
</tr>
</tbody>
</table>
TABLE 2.6

Expected number of busy memories in one cycle:

Exponential distribution for $t_p$

Constant $t_w - t_b = E[t_p]$

Simulation results

<table>
<thead>
<tr>
<th>$m$</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n=2$</td>
<td>1.4088</td>
<td>1.5931</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=3$</td>
<td>1.6185</td>
<td>1.9878</td>
<td>2.2075</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=4$</td>
<td>2.2198</td>
<td>2.5643</td>
<td>2.8004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=5$</td>
<td>2.7980</td>
<td>3.1472</td>
<td>3.4300</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=6$</td>
<td>3.4088</td>
<td>3.7122</td>
<td>4.0040</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=7$</td>
<td>3.9990</td>
<td>4.3196</td>
<td>4.5804</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$n=8$</td>
<td>4.5666</td>
<td>4.9028</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 2: Multiprocessors with $tp=tw$

2.7 Concluding Remarks

time, with mean equal to $tw$.

\[ \text{i.e. } \Pr\{tp=x\} = \lambda \exp(-\lambda x) \quad \text{where } \lambda = 1/tw = 1/\text{ta} = 1/E[tp] \]

Note that the values in Table 2.6 lie between those predicted by Strecker and Jackson, and within 5\% of the exact discrete Markov chain model for most cases. Thus, modeling the variable processing time by a constant equal to the mean processing time is a reasonable simplification. Table 2.7 shows the characteristics of the parameters in the various models.

It is important to note that with $tp=tw$ a Pc is fast enough to make a new request to memory when the memory recovers. Thus, for a 1x1 system the memory is always busy. Also, with $m \geq n$, if there is no contention for memory the maximum number of busy memories is $\min(m,n)$. An important result observed was the absence of a law of diminishing returns: the performance of a multiprocessor system with $n$ processors and $n$ memories continues to rise at a constant rate as $n$ increases. A simple exponential server model showed this rate to be 0.5; a constant processing time model predicted a slope of 0.586 for the average number of busy Mp's. The exponential server model gives the average number of busy Mp's as $n^2m/(n+m-1)$. An approximate result for constant processing times gives the average number of busy Mp's as $i^2/[1-(1-i)/j]$, where $i=\max(n,m)$ and $j=\min(n,m)$. An intuitively obvious conclusion limits the maximum number of active Pc's by $\min(n,m)$. This maximum is reached if each processor accesses only one memory all the time.
<table>
<thead>
<tr>
<th>Method</th>
<th>Processing Time</th>
<th>Memory Cycle Time</th>
<th>Analysis</th>
<th>Computational Ease</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discrete Markov Chain</td>
<td>Constant tp=tw</td>
<td>Constant</td>
<td>Exact</td>
<td>Solution is algorithmic. Unwieldy for large n.</td>
</tr>
<tr>
<td>Strecker's Approximation</td>
<td>Constant</td>
<td>Constant</td>
<td>Approximate</td>
<td>Closed form solution. Simple formula.</td>
</tr>
<tr>
<td>Continuous Time Markov Chain</td>
<td>Exponential</td>
<td>Exponential</td>
<td>Exact</td>
<td>Closed form solution. Simple formula.</td>
</tr>
<tr>
<td>Diffusion Approximation</td>
<td>Constant</td>
<td>Constant</td>
<td>Approximate</td>
<td>Closed form solution. Simple formula.</td>
</tr>
<tr>
<td>Simulation Model</td>
<td>Exponential E(tp)=tomega</td>
<td>Constant</td>
<td>Approximate</td>
<td>Unwieldy due to slow stochastic convergence.</td>
</tr>
</tbody>
</table>
CHAPTER 3
MULTIPROCESSOR SYSTEMS WITH TP>TW

In this chapter multiprocessors with tp>tw will be discussed. First, a discrete Markov chain model for a constant processing time equal to tw+tc will be developed and a general methodology for constant tp=tw+tc will be presented. A more general model for processing time having a geometric distribution with its mean value greater than tw will be described. An exponential server model developed by McCredie[McCrJ73] will also be discussed.

3.1 DISCRETE MARKOV CHAIN MODELS FOR MULTIPROCESSORS WITH TP=TW+TC

In this section, discrete markov chain models will be developed for multiprocessor system in which the processing time tp is a constant and is exactly equal to tw+tc. The timing of a typical instruction is shown below.

Note that a Pc that is serviced during this cycle operates on its data during the next memory cycle. This will be modeled by associating a server with each Pc
Figure 3.1 Queueing model for multiprocessors with $t_p = t_w + t_c$
with a constant processing time equal to $tc$; and this portion of the processing time will be called its effective execution time. As before, all processors will be assumed to be identical with $P_{ij}=1/m$. The processor servers will be lumped together as a multi-server station.

Now, if the number of Pcs is $n$ and the number Mps is $m$, the state of the queueing system shown in Fig. 3.1 can be described by a $(m+1)$-tuple $(k_0; k_1, k_2, ..., k_m)$, where $k_0$ is the number of Pcs in execute state and $k_i, 1 \leq i \leq m$, is the number of Pcs queued for Mp[i]. Since all the servers contribute fixed delays equal to $tc$ all events (entities leaving and entering queues) occur at epochs separated by integer multiples of $tc$. In other words, the system behaves as if it were clocked at intervals of $tc$. Therefore, time between significant events can be considered to advance in discrete steps. Note that an equivalent system in which $tw'=0, ta'=tc$ and $tp'=tc$ can also be represented by the model shown in Fig. 3.1.

Let $(k_0; k_1, k_2, ..., k_m)$ be a representative state i.e. it denotes all the states that can be obtained by permuting $(k_1, k_2, ..., k_m)$. Further, let $k_1, k_2, ..., k_m$ be arranged in non-increasing order. Since the number of Pcs is fixed $\sum_{i=0}^{m} k_i=n$. The state space can be divided into $n+1$ sub-spaces corresponding to integer values of $k_0$ ranging from 0 to $n$. The number of states in the 0-th sub-space is equal to the number of ways of partitioning the integer $n$ into $m$ parts. In general, the $i$-th sub-space consists of states corresponding to the partitions of $n-i$ into $m$ parts.
Initial State

2; 2,0,0,0

Reduced Partial State

1; 1,0,0,0

1; 1,0,0,0

1; 2,0,0,0

3; 1,1,0,0

1; 2,1,0,0

1; 3,0,0,0

Add 1 Fc

Add 1 more Fc

Figure 3.2 A typical enumeration tree for initial state 2; 2,0,0,0
Consider a state vector in the i-th sub-space. The value of $k_0$ is i. Let $d$ denote the number of non-zero parts of the memory-state-vector $(k_1, k_2, ..., k_m)$. Then at the end of the current memory cycle i Pcs make a new request and $d$ Pcs enter the execute state, i.e. during the next cycle the state of the system is located in the $d$-th sub-space. State transitions can be described by an enumeration tree similar to that used in chapter 2, Fig. 2.3. Figure 3.2 transitions from the state $(2; 2,0,0,0)$ for a 4x4 multiprocessor system. Such trees can now be constructed for each state and the transition matrix evaluated. In Fig. 3.2, there is 1 way of reaching $(1; 3,0,0,0)$, $(3+3*2)$ i.e. 9 ways of reaching $(1; 2,1,0,0)$ and $3*2$ i.e. 6 ways of reaching $(1; 1,1,1,0)$. Once the entire transition matrix is generated the stationary state probabilities can be obtained.

The number of states of this discrete Markov chain model increases faster than the exact Markov chain model for $tp=tw$, described in chapter 2. Table 3.1 compares the number of states for the two models. An approximate Markov chain model will now be proposed. The system behavior will be modified to simplify the analysis. At the end of each cycle the active Pcs (those that are served by memory during the current cycle) will enter the execute state. However, all the queued Pcs will be removed from the memory queues and will be allowed to make new requests to memory along with those Pcs that were in execute state during the current cycle. Let the current state be $(k_0, k_1, k_2, ..., k_m)$, with exactly $d$ Mp queues occupied. Then, at the end of the current cycle, $d$ Pcs enter the
TABLE 3.1

Comparison of the Number of States for Discrete Markov Chain Models for $t_p = t_w$ and $t_p = t_w + t_c$

<table>
<thead>
<tr>
<th>$n=m$</th>
<th>$t_p = t_w$</th>
<th>$t_p = t_w + t_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>15</td>
<td>44</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>66</td>
</tr>
<tr>
<td>9</td>
<td>30</td>
<td>96</td>
</tr>
<tr>
<td>10</td>
<td>42</td>
<td>138</td>
</tr>
<tr>
<td>11</td>
<td>56</td>
<td>194</td>
</tr>
<tr>
<td>12</td>
<td>77</td>
<td>271</td>
</tr>
</tbody>
</table>
execute state and \( n-d \) Pc's make new requests. Thus, for this simplified model, the state of the system is characterized by the number of Pc's that make new requests at the beginning of a cycle.

Now, if \( i \) Pc's make a new request to \( m \) Mp's, the probability that \( Mp[i] \) gets at least one request is \( 1-(1-1/m)^i \). With \( i \) Pc's accessing \( m \) Mp's simultaneously, the number of busy Mp's can take any integer value from 1 up to \( \min(i,m) \). The probability of exactly \( j \) Mp's being occupied is given by the ratio of the number of ways that \( j \) Mp's can be occupied and \( i-j \) Mp's not be occupied to the total number of ways of assigning \( i \) Pc's among \( m \) Mp's i.e. \( m^i \). In section 2.3, it was stated that the number of ways that exactly \( j \) out of \( m \) Mp's can be occupied by \( i \) Pc's is \( CM(m,i) \ast S(i,j) \). Now, if \( j \) Mp's are occupied during the current cycle then \( n-j \) Pc's make a request to Mp during the next cycle. Therefore, given that \( i \) Pc's made a request to \( M_p \) at the beginning of the current cycle, the conditional probability that \( n-j \) Pc's will make a new request at the beginning of the next cycle (which is also the end of the current cycle) is

\[
CM(m,i) \ast S(i,j)/m^i
\]

Note that the above expression denotes the probability of a transition from a current state \( i \) to a next state \( n-j \). The value of \( j \) can range from 1 to \( \min(i,m) \).

For a multiprocessor system with \( n \) Pc's and \( m \) Mp's, let \( k=\min(m,n) \). The number of occupied memories in a cycle ranges from 0 to \( k \). Therefore, the number
TABLE 3.2

Transition matrix for a 4x4 system with \( t_p = t_w + t_c \)

\[
\begin{pmatrix}
X_0 \\
X_1 \\
X_2 \\
X_3 \\
X_4
\end{pmatrix} = 
\begin{pmatrix}
0.0000 & 0.0000 & 0.0000 & 0.0000 & 0.09375 \\
0.0000 & 0.0000 & 0.0000 & 0.3750 & 0.56250 \\
0.0000 & 0.0000 & 0.7500 & 0.5625 & 0.32813 \\
0.0000 & 1.0000 & 0.2500 & 0.0625 & 0.01563 \\
1.0000 & 0.0000 & 0.0000 & 0.0000 & 0.0000
\end{pmatrix} \times 
\begin{pmatrix}
X_0 \\
X_1 \\
X_2 \\
X_3 \\
X_4
\end{pmatrix}
\]

\( X_0 + X_1 + X_2 + X_3 + X_4 = 1 \)
Figure 3.3a  Execution Rate as a fraction of the number of Pc's
Figure 3.3b  Execution rate as a fraction of the number of Mp's
Chapter 3: Multiprocessors with $tp=tw$

3.1 Discrete Markov Chain Model for $tp=tw+tc$

of Pcs's that can be assigned at the beginning of a cycle ranges from $n-k$ to $n$.

The approximate model has $k+1$ states. Let $X_{n-k}, X_{n-k+1}, \ldots, X_n$ denote the steady state probabilities. Then, the execution rate is

$$\sum_{i=n-k}^{n} X_i \cdot [1-(1-1/m)^i] \cdot m$$

Table 3.2 shows the transition matrix for a 4x4 multiprocessor system. Appendix A-5 contains a listing of a FORTRAN program that computes the steady state probabilities and hence the execution rate for a $nxm$ system. Figure 3.3 depicts plots of the execution rate as a function of $m$ and $n$, obtained from the approximate Markov chain model. Table 3.3 compares the analytic results with a 90\% confidence interval obtained by a Monte Carlo simulation of the exact system behavior. The simulation consisted of 10 independent experiments of length equal to 4000 cycles.

3.1.1 General Technique for Constant $tp=tw+istc$

In general, if the processing time is a constant and equal to $tw+istc$, the instruction timing diagram is as shown in Fig. 3.4a. In this case, the execution phase is $i$ cycles long. This can be modeled by an $i$-stage server shown in Fig. 3.4b. At any given time in the execute phase a Pc is in one of the $i$ stages; advancing one stage every cycle. Now, the system state can be represented by $(j_1, j_2, \ldots, j_i; k_1, k_2, \ldots, k_m)$, where $j_i$ is the number of Pcs's in the execute-stage $i$, and the $k_1$'s denote the memory queue sizes. As before, let $d$ denote the number of non-zero $k_1$'s. Then, at the beginning of the next cycle, $d$
Figure 3.4a  Instruction timing diagram for $tp = tw + i*tc$

Figure 3.4b  $i$-stage server model for a $Pc$
TABLE 3.3

Average Number of Busy M's

\( tp = tw + tc \)

<table>
<thead>
<tr>
<th>( n \times m )</th>
<th>90% confidence interval from simulation</th>
<th>Analytic Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2 \times 2 )</td>
<td>(1.0000, 1.0000)</td>
<td>1.0000</td>
</tr>
<tr>
<td>( 2 \times 4 )</td>
<td>(1.0000, 1.0000)</td>
<td>1.0000</td>
</tr>
<tr>
<td>( 2 \times 8 )</td>
<td>(1.0000, 1.0000)</td>
<td>1.0000</td>
</tr>
<tr>
<td>( 2 \times 16 )</td>
<td>(1.0000, 1.0000)</td>
<td>1.0000</td>
</tr>
<tr>
<td>( 4 \times 2 )</td>
<td>(1.5647, 1.5843)</td>
<td>1.6000</td>
</tr>
<tr>
<td>( 4 \times 4 )</td>
<td>(1.8194, 1.8276)</td>
<td>1.8276</td>
</tr>
<tr>
<td>( 4 \times 8 )</td>
<td>(1.9126, 1.9244)</td>
<td>1.9197</td>
</tr>
<tr>
<td>( 4 \times 16 )</td>
<td>(1.9418, 1.9817)</td>
<td>1.9612</td>
</tr>
<tr>
<td>( 8 \times 2 )</td>
<td>(1.8251, 1.8547)</td>
<td>1.9692</td>
</tr>
<tr>
<td>( 8 \times 4 )</td>
<td>(2.8460, 2.9206)</td>
<td>3.8259</td>
</tr>
<tr>
<td>( 8 \times 8 )</td>
<td>(3.4858, 3.5346)</td>
<td>3.5530</td>
</tr>
<tr>
<td>( 16 \times 2 )</td>
<td>(1.9086, 1.9404)</td>
<td>1.9998</td>
</tr>
<tr>
<td>( 16 \times 4 )</td>
<td>(3.4936, 3.5677)</td>
<td>3.8772</td>
</tr>
<tr>
<td>( 16 \times 8 )</td>
<td>(5.4431, 5.6254)</td>
<td>5.9053</td>
</tr>
<tr>
<td>( 16 \times 16 )</td>
<td>(6.8140, 6.9788)</td>
<td>7.0136</td>
</tr>
</tbody>
</table>
3.2 DISCRETE MARKOV CHAIN MODELS FOR GEOMETRICALLY DISTRIBUTED TP

For our next model, let the processing time have a geometric distribution given by,

\[ \text{Prob}[tp=tw+i*tc] = \beta^i \alpha^1 \]

where \( \beta = 1 - \alpha \)

Then, the mean processing time is given by,

\[ \sum_{i=0}^{\infty} \beta^i \alpha^1 (tw+i*tc) \]

i.e. \( tw + \beta \sum_{i=0}^{\infty} i \alpha^1 \)

i.e. \( tw + tc \beta \alpha \sum_{i=0}^{\infty} i \alpha^1 \)

i.e. \( tw + tc \beta \alpha / (1-\alpha) \)

i.e. \( tw + \alpha tc / (1-\alpha) \)

Thus any mean value of \( tp \) greater than \( tw \) can be modeled by appropriately choosing \( \alpha \). This model is more general than the models of the previous section.
Figure 3.5 Structure of the queuing model for geometrically distributed processing time.
Also, a single model with \( \alpha \) as a parameter handles all cases where the mean value of \( tp \) is greater than \( tw \). The geometric distribution is a discrete analog of the exponential distribution. The measurements reported in chapter 5 show that the processing time distribution is indeed close to a shifted exponential.

Once again, the representative state of an exact discrete Markov chain model is given by the vector \((k_0, k_0, k_0, \ldots, k_0)\). Also, \( \sum_{i=0}^{m} k_i = n \) and \( k_i \) is the number of Pcs queued for \( M_p[i] \), \( 1 \leq i \leq m \), and \( k_e \) is the number of Pcs in execute state. The reduced partial state at the end of the current cycle is given by the vector \((0, j_1, j_2, \ldots, j_m)\), where \( j_i = \max(0, k_i - 1) \) for \( 1 \leq i \leq m \). Note that \( \sum_{i=1}^{m} j_i = n - k_e - d \), where \( d \) is the number of non-zero \( k_i \)'s. Now, up to \( k_e + d \) Pcs are potentially available to be reassigned to the various queues. Let the next state be denoted by \((l_0, l_1, l_2, \ldots, l_m)\). Figure 3.5 shows the structure of the queueing model. Each Pc has a probability \( \alpha \) of going back into the execute phase. Since \( l_0 \) denotes the number of Pcs that are in the execute phase during the next cycle, the range of \( l_0 \) is from 0 to \( k_e + d \). The value of \( l_0 \) is governed by the following probability function,

\[
\text{Prob}(l_0 = i) = \binom{k_0 + d}{i} \times \alpha^i \times \beta^{k_0 + d - i} \times \text{SiSk}_0 + d
\]

The rest of the next state vector can be determined by using an enumeration tree. Figure 3.6 shows a typical enumeration tree for an initial state of \((2, 2, 0, 0, 0, 0)\).

Once again, to reduce the number of states let us make the following approximation. At the end of an \( M_p \) cycle, those Pcs's that were in the \( M_p \) queues
Figure 3.6: A typical enumeration tree

* indicates a next state that can be reached from the initial state 2; 0, 0, 0.

Initial state

2; 2, 0, 0, 0, 0

1; 1, 0, 0, 0, 0

0; 1, 0, 0, 0, 0

0; 0, 1, 0, 0, 0

1; 0, 0, 1, 0, 0

1; 0, 1, 0, 1, 0

1; 0, 1, 1, 0, 1

1; 0, 1, 1, 1, 1

1; 0, 2, 0, 0, 0

1; 0, 2, 1, 0, 0

1; 1, 0, 0, 0, 0

1; 1, 0, 1, 0, 0

1; 1, 1, 0, 0, 0

1; 1, 1, 1, 0, 0

1; 1, 1, 1, 1, 1

1; 1, 2, 0, 0, 0

1; 1, 2, 1, 0, 0

1; 1, 2, 1, 1, 0

1; 1, 2, 1, 1, 1

1; 1, 2, 2, 0, 0

1; 1, 2, 2, 1, 0

1; 1, 2, 2, 1, 1

1; 1, 2, 2, 1, 1

1; 1, 2, 2, 2, 0

1; 1, 2, 2, 2, 1

1; 1, 2, 2, 2, 1

1; 1, 2, 2, 2, 1

1; 1, 2, 2, 2, 1

1; 1, 2, 2, 2, 1

1; 1, 2, 2, 2, 1
Chapter 3: Multiprocessors with \( t_p > tw \)

### 3.2 Discrete Markov Chain Model for Geometrically Distributed \( t_p \)

During the current cycle but not serviced are removed from the queues. They are then reassigned to the \( Mp \) queues. Now, the state is characterized by the number of \( Pc \)'s that make a request i.e. the number of \( Pc \)'s that are queued during the cycle. Thus, the number of states is \( n+1 \), viz. 0,1,2,..,\( n \). Let the number of \( Pc \)'s queued during the current cycle be \( i \). This means that the other \( n-i \) \( Pc \)'s are executing. If the \( i \) requests at the beginning of the current cycle result in \( d(\min(i,m)) \) busy \( Mp \)'s during the current cycle, then \( i-d \) \( Pc \)'s are left unserviced. Therefore, during the next cycle at least \( i-d \) \( Pc \)'s are queued for \( Mp \) service. Besides, each of the other \( n-i+d \) \( Pc \)'s has a probability \( \beta \) of making a request to memory.

The probability that \( i-d \) \( Pc \)'s are left in the \( Mp \) queues at the end of the current cycle is given by

\[
CM(m,d) \times S(i,d)/m^i
\]

The above expression ensues from the fact that \( i \) \( Pc \)'s make random requests to \( m \) \( Mp \)'s resulting in exactly \( d \) \( Mp \)'s being occupied. Also, the probability that \( j \) out of the other \( n-i+d \) \( Pc \)'s make a request at the beginning of the next cycle is

\[
\left( \frac{n-i+d}{j} \right) \times \beta^j \times \alpha^{n-i+d-j}
\]

Therefore, the probability that \( i-d+j \) \( Pc \)'s make a request to \( Mp \) during the next cycle, i.e. probability that the next state is \( i-d+j \) is

\[
CM(m,d) \times S(i,d)/m^i \times \left( \frac{n-i+d}{j} \right) \times \beta^j \times \alpha^{n-i+d-j}
\]

Table 3.4 summarizes some of the numerical results obtained from the approximate model. A listing of the FORTRAN program for this model is included in Appendix.
TABLE 3.4

Average number of busy M's

<table>
<thead>
<tr>
<th>n x m</th>
<th>α</th>
<th>95% confidence interval from simulation</th>
<th>Analytic Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x 2</td>
<td>0.1</td>
<td>(0.9130 , 0.9587)</td>
<td>1.8478</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td></td>
<td>1.7849</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td></td>
<td>1.5423</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td></td>
<td>0.9408</td>
</tr>
<tr>
<td>4 x 4</td>
<td>0.1</td>
<td>(2.2875 , 2.3258)</td>
<td>2.6079</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>(1.7382 , 1.8022)</td>
<td>2.3679</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td></td>
<td>1.7920</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td></td>
<td>0.9739</td>
</tr>
<tr>
<td>4 x 8</td>
<td>0.1</td>
<td>(2.6607 , 2.6831)</td>
<td>3.0773</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>(2.7675 , 2.8254)</td>
<td>2.6824</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td></td>
<td>1.9812</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td></td>
<td>0.9877</td>
</tr>
<tr>
<td>8 x 4</td>
<td>0.1</td>
<td>(4.3508 , 4.4713)</td>
<td>5.0243</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>(3.4158 , 3.5118)</td>
<td>4.5904</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td></td>
<td>3.5224</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td></td>
<td>1.9338</td>
</tr>
<tr>
<td>8 x 8</td>
<td>0.1</td>
<td>(5.3855 , 5.5085)</td>
<td>6.9461</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td>(6.7643 , 6.9240)</td>
<td>6.9848</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td></td>
<td>3.8693</td>
</tr>
<tr>
<td>16 x 16</td>
<td>0.1</td>
<td></td>
<td>9.8718</td>
</tr>
<tr>
<td></td>
<td>0.25</td>
<td></td>
<td>9.8441</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>(6.7643 , 6.9240)</td>
<td>6.9848</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td></td>
<td>3.8693</td>
</tr>
</tbody>
</table>
3.2 Discrete Markov Chain Model for Geometrically Distributed \( t_p \)

A-6. This model will be used in chapter 5 to predict the performance of C.mmp. For \( t_p > t_w \) this model is fairly realistic: the processing time is modeled as a geometrically distributed random variable and the memory cycle time is a constant.

3.2.1 Extensions

The basic geometric model can be used to model systems in which each P.c has a private cache memory (M.c). Let the access time of the cache be \( t_f \) (stc); and let \( x \) denote the probability of making an access to the cache. Therefore, a fraction \((1-x)\) of all memory requests is to M.p. In the queueing model shown in Fig. 3.7, \( \alpha \) denotes the probability that the execution phase goes through another cycle. Thus, \( \beta = 1-\alpha \) is the probability of a P.c finishing execution and making a request to memory. Thus, \( \beta_1 = \beta(1-x) \) and \( \beta_2 = \beta x \).

As before, 

\[
\text{Prob}\{t_p = t_w + it_c\} = \beta_1 \times \alpha^i \quad \text{for } i=0,1,2,...
\]

and,

\[
\text{Prob}\{t_p = t_c - tf + it_c\} = \beta_2 \times \alpha^i \quad \text{for } i=0,1,2,...
\]

The expected value of \( t_p \), given that all accesses are to M.p, is given by

\[
E[t_p | M_p] = t_w + t_c \cdot \beta_1 \times \alpha^i / (1-\alpha)^2
\]

Similarly,

\[
E[t_p | M_c] = t_c - t_f + t_c \cdot \beta_2 \times \alpha^i / (1-\alpha)^2
\]

Hence, the unconditional expected value of \( t_p \) is
Figure 3.7 Queueing model for multiprocessors with cache memory
3.2 Discrete Markov Chain Model for Geometrically Distributed $tp$


$$= (1-x)[tw + tc(1-x)*\alpha/(1-\alpha)] + x[(tc-tf + tc*\alpha)/(1-\alpha)]$$

If $E[tp]$ is known, $\alpha$ can be computed from the above equation. This model is fairly general in that the parameters $x$ and $E[tp]$, $tc$, $tw$ and $tf$ can be chosen arbitrarily, subject to $E[tp] \geq tw + x*(tc-tf-tw)$ and $tf < tc$. The distribution of $tp$ is now the sum of two geometrics.

Note that this model, as viewed by the $Mp$, behaves exactly like the model for geometrically distributed $tp$ shown in Fig. 3.5. The $\beta$ in Fig. 3.5 is equivalent to the $\beta$, in fig 3.7, and the $\alpha$ in Fig. 3.5 is equivalent to the $\alpha + \beta_2$ in fig 3.7. Let $R$ denote the average number of busy $Mp$'s for the model of Fig. 3.5, but with $\beta = \beta_1$ and $\alpha = 1-\beta_1$. The values of $tw$ and $tc$ is kept unchanged. Now, in the system with the cache, the average number of busy $Mp$'s is also $R$. Note that the expected value of $tp$ is different for the two cases. Now, by definition, for every $1-x$ accesses to $Mp$ there are $x$ accesses to $Mc$. Consider an interval equal to $T$ $Mp$ cycles. During this interval there are $R*T$ busy $Mp$ cycles. Hence, there are $R*T*x/(1-x)$ accesses to $Mc$. Therefore, the number of instructions executed in one $Mp$ cycle is

$$R + R*x/(1-x)$$

i.e. $R/(1-x)$

Thus, the average number of equivalent $Mp$ cycles is $R/(1-x)$.

For example, let us compute the average number of effective busy $Mp$ cycles for $ta=tw=5$, $tc=10$, $tf=1$, $x=0.75$, $E[tp]=15$, and $n=m=8$. The equation for $E[tp]$ in
Figure 3.8 Structure of McCredie's Memory Interference Model
terms of $\alpha$ gives $\alpha = 28/53$. Hence $\beta = (1-\alpha)/(1-\alpha) = 25*25/53$. Using $\beta = 0.25*25/53$ for the model of section 3.2, we get $R = 0.9371$. Hence, the average number of effective busy Mp cycles is $0.9371/0.25 = 3.7484$.

### 3.3 McCREDIE's EXPONENTIAL SERVER MODEL

As mentioned in Chapter 2, if the memory cycle time and the processing time are assumed to be exponential, the queueing results of Jackson[Jack63] can be used. The structure of McCredie's memory interference model[McCr73] is depicted in Fig. 3.8. In terms of the notation used in this chapter, $\lambda = 1/(tp-tw)$ (if $\alpha = 0$), $u = 1/tc$ and $\alpha$ is the probability of accessing a cache memory whose access time is assumed to be negligible. The time from the completion of one reference to main memory until the next access is exponentially distributed with mean $1/(\lambda*\beta)$. The model allows the first Mp module to have a different cycle time $1/v$. The probability that a request to main memory is to the first Mp module is $f$; the requests to the other Mp modules being uniformly distributed. Let $k$ be the number of Pc's queued. Using Jackson's formulae and some clever grouping of terms, the execution rate is obtained as

$$\sum_{k=0}^{n} (n-k)*\lambda*P(k)$$

---

†The reader is referred to McCr73 for the details of the derivation.
Figure 3.9 Performance predicted by McCredie's model
Chapter 3: Multiprocessors with \( tp > tw \)

3.3 McCredie's Exponential Server Model

\[
\begin{align*}
\text{where } P(k) &= \frac{n! \left( \beta \lambda \right)^k \sum_{i=0}^{K} \binom{m-2+i}{i} \left( \frac{f}{v} \right)^{k-i} \left( \frac{1-f}{v} \right)^{i} \left( \frac{1}{(m-1)u} \right)^i}{(n-k)! \sum_{k=0}^{n-k} W(k) \cdot T(k)} \\
\text{and } W(k) &= (\beta \lambda)^k \cdot n!/(n-k)! \\
\text{and } T(k) &= \sum_{j=0}^{K} \binom{m-2+j}{j} \left( \frac{f}{v} \right)^{k-j} \left( \frac{1-f}{(m-1)u} \right)^j
\end{align*}
\]

Figure 3.9 shows some of the results predicted by the model. The advantage of this model is that it allows a cache memory and an \( M_p \) module with a different speed and a different access probability. Note that with the presence of the cache \( \lambda = 1/(tp-tw) \); but \( \lambda = 1/(tp-\beta tw) \).

3.4 STRECKER'S ANALYSIS

This section will briefly review Strecker's analysis of multiprocessors with \( tp > tw \)[StreW70]. The processing time is assumed to be a constant. The number of processors queued is, in general, less than \( n \). The probability a \( P_c \) is queued is denoted by \( p_m \). Then, assuming a binomial distribution for the queued processors, the execution rate is

\[
m * [1 - (1 - p_m / m)^n]
\]

Now, because the number of \( P_c \)'s queued is binomially distributed, the average number of \( P_c \)'s queued is \( n * p_m \). Hence,
Chapter 3: Multiprocessors with \( \text{tp}>\text{tw} \)

3.4 Strecker’s Analysis

\[ p_m = \frac{\text{average number of Pc’s queued}}{n} \]

Strecker’s flow diagram of the instruction execution is shown below.

\[ \text{output rate} \ (m/\text{tc}) \times [1 - (1-p_m/m)^n] \]

![Flow diagram](image)

Strecker states that the average number of Pc’s not queued is the product of the average unit execution rate and the effective Pc delay \( \text{tp}-\text{tw} \). Thus,

\[ p_m = 1 - \frac{m}{n/\text{tc}} \times (1-(1-p_m/m)^n) \times (\text{tp}-\text{tw}) \]

which is a \( n \)-th order polynomial equation in \( p_m \) which has only one solution for \( p_m \) in the interval \((0,1)\). This value of \( p_m \) is used in the earlier expression for the execution rate. This model is fairly simple and good for larger values of \( \text{tp} \).

\[ \uparrow \text{This follows directly from Little’s formula, } L=\lambda \times W \ [\text{cf. Litt61}]. \]
Chapter 3: Multiprocessors with $tp>tw$

3.5 The Effect of Cache on Systems with $tp>tw$

3.5 THE EFFECT OF CACHE ON SYSTEMS WITH $tp>tw$

One of the techniques used to increase the execution rate of uniprocessor systems involves the use of a fast cache memory, $Mc$. In multiprocessor systems, the cache not only provides a memory with a smaller access time but also reduces the traffic through the crosspoint switch. This reduces the memory interference and reduces the amount of time the processor spends waiting for $Mp$ service. In fact, the use of private caches for the $Pc$'s is being considered for CMU's C.mmp.

In this section, we shall characterize the processing time by a single constant value $tp$, with $Pr_{i,j}=1/m$. Let the cache have an access time $tf$ (slta) and a rewrite time $tr$ (stw). Since $tp>tw$ and $tr<tw$, the cache always recovers before the $Pc$ can make its next request. Let $\alpha$ be the probability of accessing $Mc$ i.e. the probability that the cache contains the information needed by the $Pc$. The probability of accessing $Mp$ is $\beta=1-\alpha$.

Now, the time needed to execute one unit instruction out of cache is $Wc=tp+tf$. Let $Wm$ be the average time needed to execute one unit instruction out of $Mp$. Hence, the average time needed to execute one unit instruction is

$$W_{avg} = \alpha Wc + \beta Wm$$

Therefore, the execution rate is $n/W_{avg}$.

The problem now reduces to evaluating $Wm$. Let us focus our attention on a
single Mp unit. Further, let us assume that the number of queued Pc's for that
Mp unit follows the binomial distribution; and let \( p_m \) be probability that a Pc
is queued for one of the Mp's. Hence, the probability of being queued for the Mp
unit under consideration is \( p_m/m \), since all Mp's have the same speed and
\( P_1 = 1/m \). From the binomial distribution for the queued Pc's, it follows that, \( L \),
the average number of Pc's queued for the Mp is \( n_p_m/m \). The rate \( \lambda \) at which Pc's
are served by this Mp is \( m(1-(1-p_m/m)^n)/tc \). Using Little's formula, \( L = \lambda W \), we
obtain the average waiting time for an Mp as
\[
W_m = \frac{n_p_m * tc}{(1-(1-p_m/m)^n)}
\]
Therefore \( W_m \), the average time for one instruction out of Mp, is
\[
W_{avg} = \alpha * (tp-tw) + \beta * \frac{n_p_m * tc}{(1-(1-p_m/m)^n)}
\]
Hence,
\[
W_{avg} = \alpha * (tp-tw) + \beta * \frac{n_p_m * tc}{[1-(1-p_m/m)^n]}
\]
Now, the only undetermined quantity is \( p_m \). In \( T \) Mp cycles the total number
of busy Mp cycles is
\[
T_1 = m * [1-(1-p_m/m)^n] * T
\]
Hence, the total number of busy cache cycles is
\[
T_2 = \alpha / \beta * [m * [1-(1-p_m/m)^n] * T]
\]
and the total number of unit instructions is the sum of the above two
expressions, i.e. \( T_1 + T_2 \). Therefore, the unit execution rate is
\[
m / (\beta * tc) / [1-(1-p_m/m)^n]
\]
Recall that we had earlier found the execution rate to be \( n/W_{avg} \). Equating the
two and rearranging the terms we get
Chapter 3: Multiprocessors with $tp>tw$

3.5 The Effect of Cache on Systems with $tp>tw$

\[ p_m = 1 - \frac{m}{n\beta tc} \times \left[ \alpha \frac{(tp+tf)}{tp-tw} + \beta (tp-tw) \right] \times \left[ 1 - \left( 1 - \frac{p_m}{m} \right) ^m \right] \]

The above equation can be solved iteratively for $p$. A FORTRAN program that computes the execution rate for this model is listed in Appendix A-7. Figure 3.10 shows the execution rate of an 8x8 and a 16x16 multiprocessor system for various values of the other parameters. Some simulation confidence intervals are also depicted. Note that with $\alpha=0$, this model yields the same results as Strecker's model described in Section 3.4. The major advantage of this model is that it allows the cache speed to be a control parameter of the analysis.

3.6 CONCLUDING REMARKS

In general, if the processing time is greater than the memory rewrite time, then in the absence of memory contention one instruction is executed by each $Pc$ every $ta+tp$ time units. Thus, the theoretical maximum value of the average number of busy $Mp$'s is $n*tc/(ta+tp)$. Hence, if $m>n$, even in the absence of memory interference, the $Mp$'s will have idle periods. Comparing Fig. 2.11 and Fig. 3.3a, the curves for $tp>tw$ reach their asymptotic maximum for smaller values of $m$. Systems with $tp>tw$ are generally processor speed limited and relatively small performance improvement will be obtained if the memory speed is increased.
Figure 3.10a The Effect of Cache on a 8x8 system

Probability of accessing cache

Unit Execution Rate

n=6  m=6
tc=10  ta=tw=5
tf=1, tp=5
tf=1, tp=10
tf=2, tp=5
Figure 3.101 The Effect of Cache on 16x8 systems
In this chapter, an approximate model is proposed for multiprocessor systems with \( t_p < t_w \) and \( P_{1,j} = 1/m \). The processing time will be assumed to be a constant. In this case, a classical queueing model is made difficult due to the following reason. In conventional queueing models, the service center can start serving a new customer as soon as the last customer leaves. For core memories, due to the rewrite time, the memory has to wait. However, for \( t_p \geq t_w \), this problem can be surmounted by delaying the \( P_c \) in the \( M_p \) queue for an additional time equal to \( t_w \) and reducing the effective execution time by the same amount. Hence, with this modification, the \( M_p \) can start serving the next \( P_c \) in its queue when the current \( P_c \) leaves the queue. But, with \( t_p < t_w \), the \( P_c \) can make a new request before the memory that served it last recovers. Strictly speaking, a discrete Markov chain model can be formulated by making the basic time interval equal to the highest common factor of \( t_p, t_w \), and \( t_a \). However, the number of possible epochs in a memory cycle and the size of the state vector and the number of states is very large.
4.1 AN APPROXIMATE MODEL FOR TP< TW

The results of the discrete Markov chain model of section 2.2 will be used to obtain an approximate of the execution rate. Consider an Mp cycle in which i Mp's are initially busy. If tp=tw, the i active Pc's make a request at the end of the cycle. However, if tp<tw, an active Pc makes a new request before the Mp that served it recovers. If this request is made to an Mp that is not busy, the new request can be served immediately. Consider an Mp cycle in which i Mp's are initially busy. During this cycle i Pc's are initially active. Now, these i Pc's can receive more service if they make their next request to the m-i idle Mp's. Now, if this request is made to an idle Mp, the effective service time (not including waiting time) for the active Pc is ta+tp. However, if this request is made to a busy Mp there is no increase in the execution rate due to the fact that tp<tw. Hence, the effective service time is tc.

Let b_i denote the probability that i Mp's are busy, obtained from the exact discrete Markov chain model for tp=tw. Now, the average number of idle Mp's that receive the next request is given by

\[(m-i) \ast \left([1 - (1-1/m)]\right)\]

Thus, the probability that an active Pc gets serviced by an idle Mp is

\[(m-i) \ast \left[1 - (1-1/m)\right] / i\]

Note that the above is a conditional probability based on the fact that i Pc's are active. Therefore, the unconditional probability that the effective service
### TABLE 4.1

Average Number of Busy Mp's

<table>
<thead>
<tr>
<th>n x m</th>
<th>tp=5</th>
<th>tp=4</th>
<th>tp=3</th>
<th>tp=2</th>
<th>tp=1</th>
<th>tp=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x 2</td>
<td>1.5000</td>
<td>1.5385</td>
<td>1.5790</td>
<td>1.6216</td>
<td>1.6667</td>
<td>1.7143</td>
</tr>
<tr>
<td>2 x 4</td>
<td>1.7580</td>
<td>1.8451</td>
<td>1.9512</td>
<td>2.0782</td>
<td>2.2847</td>
<td>2.3579</td>
</tr>
<tr>
<td>2 x 8</td>
<td>1.8750</td>
<td>2.0215</td>
<td>2.1928</td>
<td>2.3958</td>
<td>2.6403</td>
<td>2.9483</td>
</tr>
<tr>
<td>2 x 16</td>
<td>1.9375</td>
<td>2.1182</td>
<td>2.3362</td>
<td>2.6041</td>
<td>2.9414</td>
<td>3.3732</td>
</tr>
<tr>
<td>4 x 2</td>
<td>1.7500</td>
<td>1.7722</td>
<td>1.7949</td>
<td>1.8182</td>
<td>1.8421</td>
<td>1.8667</td>
</tr>
<tr>
<td>4 x 4</td>
<td>2.6210</td>
<td>2.6947</td>
<td>2.7832</td>
<td>2.8721</td>
<td>2.9669</td>
<td>3.0681</td>
</tr>
<tr>
<td>4 x 8</td>
<td>3.2652</td>
<td>3.4429</td>
<td>3.6418</td>
<td>3.8633</td>
<td>4.1145</td>
<td>4.4411</td>
</tr>
<tr>
<td>4 x 16</td>
<td>3.6268</td>
<td>3.9851</td>
<td>4.2297</td>
<td>4.6131</td>
<td>5.0729</td>
<td>5.6345</td>
</tr>
<tr>
<td>8 x 2</td>
<td>1.8750</td>
<td>1.8868</td>
<td>1.8987</td>
<td>1.9108</td>
<td>1.9231</td>
<td>1.9355</td>
</tr>
<tr>
<td>8 x 4</td>
<td>3.2657</td>
<td>3.3148</td>
<td>3.3654</td>
<td>3.4176</td>
<td>3.4714</td>
<td>3.5269</td>
</tr>
<tr>
<td>8 x 8</td>
<td>4.9471</td>
<td>5.1023</td>
<td>5.2676</td>
<td>5.4439</td>
<td>5.6324</td>
<td>5.8345</td>
</tr>
<tr>
<td>8 x 16</td>
<td>6.3149</td>
<td>6.6579</td>
<td>7.0403</td>
<td>7.4692</td>
<td>7.9539</td>
<td>8.5857</td>
</tr>
<tr>
<td>16 x 2</td>
<td>1.9375</td>
<td>1.9436</td>
<td>1.9497</td>
<td>1.9558</td>
<td>1.9620</td>
<td>1.9683</td>
</tr>
<tr>
<td>16 x 4</td>
<td>3.6270</td>
<td>3.6538</td>
<td>3.6811</td>
<td>3.7088</td>
<td>3.7369</td>
<td>3.7654</td>
</tr>
</tbody>
</table>
Chapter 4: Multiprocessors with $tp<tw$

Approximate Model for $tp<tw$

Time of an active PC is $ta+tp$ is given by

$$\sum b_i * (m-i) * [1 - (1-1/m)^n] / i$$

Let $f$ be the value of the above expression. The number of active PC's is obtained from the discrete Markov chain model for $tp-tw$, denoted by $X$. The expected value of the service time for an active PC is

$$f * (ta+tp) + (1-f) * tc$$

Hence, the execution rate, expressed as unit instructions per second, is given by

$$X / [f*(ta+tp) + (1-f)*tc]$$

The average number of busy MP cycles can be obtained by multiplying the above expression by $tc$.

Table 4.1 presents the results obtained for various values of $m$, $n$, $ta$, $tp$, and $tc$. Since this model is approximate it is meaningful to compare its results with a confidence interval obtained from simulation. Table 4.2 compares the results of this model and Strecker's model [StreW70] with simulation results. The results of this model are within 5% of the simulation. Figures 4.1 and 4.2 illustrate the effects of $n$ and $m$ on the MPAR.

4.2 STRECKER’S MODEL

Strecker [StreW70] uses his approximate model for $tp-tw$ to analyze $tp<tw$. He
Figure 4.1: The effect of adding Pc's

Effective MpAR

- Number of Pc's: n
- m = 4
- m = 8
- t_e = 10
- t_r = 5
Figure 4.2 The effect of adding Mp's

$t_c = 10$

$ta = tw = 5$

Effective NPAR vs. Number of Mp's

$n = 8$

$n = 4$

$tp = 1$

$tp = 3$

$tp = 5$
Chapter 4: Multiprocessors with \( tp<tw \)

4.2 Strecker's Model

defines the probability of an active Pc making a request to an occupied Mp as
\[ p(occ) = \text{average number of occupied Mps} / m \]

The probability of a request to an unoccupied Mp is
\[ p(unocc) = 1 - p(occ) \]

Using his model for \( tp=tw \),
\[ p(occ) = 1 - (1-1/m)^n \]

Thus, the average amount of time required to execute an instruction is
\[ E[t] = p(occ) \times tc + p(unocc) \times (ta+tp) \]
\[ = tc + (1-1/m)^n \times (tp-tw) \]

Therefore, the average number of busy Mp cycles is
\[ tc \times m \times [1-(1-1/m)^n] / E[t] \]

Note that Strecker assumes a constant probability for an Mp being occupied. In the model of Sec. 4.1 the probability of an Mp being occupied depends on the number of busy Mp's during the cycle. Simulation results summarized in Table 4.2 show that the model proposed in section 4.1 is better than Strecker's model.

4.3 CONCLUDING REMARKS

With one Pc and one Mp there is no advantage gained by the fact that \( tp<tw \).

However, if an active Pc can make its next request to an unoccupied Mp
### TABLE 4.2a

Average number of busy Mp's

Comparison of Analytic Models and Simulation Results

\[ ta = tw = 5 \quad tp = 1 \quad tc = 10 \]

<table>
<thead>
<tr>
<th>( n \times m )</th>
<th>Analytic Model (Section 4.1)</th>
<th>90% confidence interval from simulation</th>
<th>Strecker's Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \times 2</td>
<td>1.6567</td>
<td>(1.5591, 1.6053)</td>
<td>1.6667</td>
</tr>
<tr>
<td>2 \times 4</td>
<td>2.2047</td>
<td>(2.1342, 2.2239)</td>
<td>2.2531</td>
</tr>
<tr>
<td>2 \times 4</td>
<td>2.6403</td>
<td>(2.6039, 2.6421)</td>
<td>2.7027</td>
</tr>
<tr>
<td>2 \times 8</td>
<td>2.6403</td>
<td>(2.6039, 2.6421)</td>
<td>2.7027</td>
</tr>
<tr>
<td>2 \times 16</td>
<td>2.9414</td>
<td>(2.9240, 2.9858)</td>
<td>2.9880</td>
</tr>
<tr>
<td>4 \times 2</td>
<td>1.8421</td>
<td>(1.7746, 1.7882)</td>
<td>1.9231</td>
</tr>
<tr>
<td>4 \times 4</td>
<td>2.9669</td>
<td>(2.8102, 2.9072)</td>
<td>3.1306</td>
</tr>
<tr>
<td>4 \times 8</td>
<td>4.1145</td>
<td>(3.9982, 4.1092)</td>
<td>4.3245</td>
</tr>
<tr>
<td>4 \times 16</td>
<td>5.8729</td>
<td>(5.8587, 5.1074)</td>
<td>5.2682</td>
</tr>
<tr>
<td>8 \times 2</td>
<td>1.9231</td>
<td>(1.8514, 1.9131)</td>
<td>1.9953</td>
</tr>
<tr>
<td>8 \times 4</td>
<td>3.4714</td>
<td>(3.2850, 3.3968)</td>
<td>3.7497</td>
</tr>
<tr>
<td>8 \times 8</td>
<td>5.6324</td>
<td>(5.3417, 5.5750)</td>
<td>6.0879</td>
</tr>
<tr>
<td>8 \times 16</td>
<td>7.9539</td>
<td>(7.7336, 7.9873)</td>
<td>8.4755</td>
</tr>
<tr>
<td>16 \times 2</td>
<td>1.9620</td>
<td>(1.9225, 1.9551)</td>
<td>2.0080</td>
</tr>
<tr>
<td>16 \times 4</td>
<td>3.7369</td>
<td>(3.6047, 3.7187)</td>
<td>3.9758</td>
</tr>
<tr>
<td>16 \times 8</td>
<td>6.7392</td>
<td>(6.4681, 6.5891)</td>
<td>7.4052</td>
</tr>
<tr>
<td>16 \times 16</td>
<td>10.969</td>
<td>(10.5279, 10.7559)</td>
<td>12.814</td>
</tr>
</tbody>
</table>
### TABLE 4.2b

Average number of busy Mp's

Comparison of Analytic Models and Simulation Results

\( ta = tw = 5 \quad tp = 3 \quad tc = 10 \)

<table>
<thead>
<tr>
<th>( n \times m )</th>
<th>Analytic Model (Section 4.1)</th>
<th>90% confidence interval from simulation</th>
<th>Strecker's Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \times 2</td>
<td>1.5790</td>
<td>(1.5368, 1.5457)</td>
<td>1.5789</td>
</tr>
<tr>
<td>2 \times 4</td>
<td>1.9512</td>
<td>(1.9058, 2.0012)</td>
<td>1.9718</td>
</tr>
<tr>
<td>2 \times 8</td>
<td>2.1928</td>
<td>(2.2019, 2.2059)</td>
<td>2.2140</td>
</tr>
<tr>
<td>2 \times 16</td>
<td>2.3362</td>
<td>(2.3441, 2.3496)</td>
<td>2.3507</td>
</tr>
<tr>
<td>4 \times 2</td>
<td>1.7949</td>
<td>(1.7623, 1.7767)</td>
<td>1.8987</td>
</tr>
<tr>
<td>4 \times 4</td>
<td>2.7832</td>
<td>(2.6950, 2.8028)</td>
<td>2.9191</td>
</tr>
<tr>
<td>4 \times 8</td>
<td>3.6410</td>
<td>(3.6021, 3.6797)</td>
<td>3.7502</td>
</tr>
<tr>
<td>4 \times 16</td>
<td>4.2297</td>
<td>(4.2319, 4.2863)</td>
<td>4.3056</td>
</tr>
<tr>
<td>8 \times 2</td>
<td>1.8987</td>
<td>(1.8530, 1.9054)</td>
<td>1.9937</td>
</tr>
<tr>
<td>8 \times 4</td>
<td>3.3654</td>
<td>(3.2646, 3.4602)</td>
<td>3.6371</td>
</tr>
<tr>
<td>8 \times 8</td>
<td>5.2676</td>
<td>(5.1470, 5.2406)</td>
<td>5.6336</td>
</tr>
<tr>
<td>8 \times 16</td>
<td>7.0403</td>
<td>(6.9457, 7.1531)</td>
<td>7.3289</td>
</tr>
<tr>
<td>16 \times 2</td>
<td>1.9497</td>
<td>(1.9212, 1.9551)</td>
<td>2.0000</td>
</tr>
<tr>
<td>16 \times 4</td>
<td>3.6811</td>
<td>(3.5241, 3.7189)</td>
<td>3.9679</td>
</tr>
<tr>
<td>16 \times 8</td>
<td>6.5204</td>
<td>(6.3928, 6.4759)</td>
<td>7.2261</td>
</tr>
<tr>
<td>16 \times 16</td>
<td>10.254</td>
<td>(9.9522, 10.350)</td>
<td>11.093</td>
</tr>
</tbody>
</table>
significant increases in the execution rate can be obtained. The theoretical maximum execution rate for each Pc is \(1/(t_a+tp)\); the average number of busy Mp cycles is limited by \(n*tc/(t_a+tp)\), if \(m\) is large enough; else it is bound by \(m\).
An important aspect in the development of mathematical models is the modeling process: the abstraction of the complex physical process to a model that is mathematically tractable. This chapter is devoted to validating the mathematical model.

One of the main parameters of the models developed in this thesis is the processing time. In order to evaluate the probability distribution and the mean of the processing time, measurements were made of the dynamic usage of the PDP-11/20 instruction set. B. Aygun's Dynamic Analysis and Measurement Environment [AyguB73] was used to simulate the execution of over 34,500 PDP-11 instructions in carefully selected main-loop portions of four programs. The PDP-11 Processor Handbook, Interface Manual and Engineering Drawings were used to obtain the instruction timing for the various instructions and addressing modes. This information can be used to predict the performance of a multiprocessor system like C.mmp, which uses the PDP-11/20 as the processor. Note that the analytic models in this thesis are general; C.mmp is used in this chapter as a typical case for validation and illustration of the use of these models.
5.1 PDP-11/20 OVERVIEW

A brief discussion of the instruction timing and formats is presented here. The PDP-11/20 processor has five major states: fetch, source, destination, execute and service. The first four states are used during normal operation; service is used during special operations, such as traps and interrupts.

Fetch: locates and decodes an instruction. When fetch is completed, the processor enters another major state, depending on the type of instruction decoded. It is possible to go from fetch to any other state, including back to fetch. Every instruction starts by first entering the fetch state.

Source: decodes the source field of a double-operand instruction and transfers the source operand to the appropriate location. The source major state is entered only if the instruction is a double-operand type.

Destination: decodes the destination field of the appropriate instruction. Destination fields are present in both single and double-operand instructions. Destination operand is accessed and transferred to the appropriate location.

Execute: uses the data obtained during previous major states to perform the
**Figure 5.1a Single Operand Instruction Format**

| OP CODE | MODE | $|$ | Rn |
|---------|------|---|-----|
| 15      | 6    | 5 | 4  |

Destination Address

* = Specifies Direct or Indirect Address  
** = Specifies How Register will be used  
*** = Specifies One of 8 General Purpose Registers

**Figure 5.1b Double Operand Instruction Format**

| OP CODE | MODE | $|$ | Rn | MODE | $|$ | Rn |
|---------|------|---|-----|------|---|-----|
| 15      | 12   | 11 | 10 | 9    | 8 | 6   |

Source Address  
Destination Address

* = Direct/Deferred Bit for Source and Destination Address  
** = Specifies How Selected Registers are to be used  
*** = Specifies a General Register
specified operation. During this state arithmetic operations, logic
functions, and tests are performed, and the destination location is updated
if required.

Service: used to execute special operations, such as interrupts, trap, etc.

Although the major states follow the sequence of fetch, source, destination,
execute, and service, not all major states are required for every instruction.
The processor enters only those major states necessary to execute the current
instruction. The minimum sequence is from a fetch of one instruction directly to
the fetch of the next instruction. The maximum sequence is fetch, source,
destination, execute, service and back to fetch. The Interface Manual contains
more detailed information about the states needed for various instructions.

The instruction format for all single operand instructions (such as clear,
increment, test) is shown in Fig. 5.1a. Operations that imply two operands (such
as add, subtract, move and compare) are handled by instructions that specify two
addresses. The first operand is called the source operand, the second the
destination operand. Bit assignments in the source and destination address
fields may specify different modes and different general registers. The
instruction format for the double operand instruction is depicted in Fig. 5.1b.
Table 5.1a summarizes the four basic modes used with direct addressing; the four
basic modes used with deferred addressing are described in Table 5.1b. The
PDP-11 Processor Handbook contains numerous illustrative examples.
## TABLE 5.1a

Direct Addressing Modes of PDP-11

<table>
<thead>
<tr>
<th>Binary Code</th>
<th>Name</th>
<th>Assembler Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Register</td>
<td>Rn</td>
<td>Register contains operand.</td>
</tr>
<tr>
<td>010</td>
<td>Autoincrement</td>
<td>(Rn)+</td>
<td>Register is used as a pointer to sequential data then incremented.</td>
</tr>
<tr>
<td>100</td>
<td>Autodecrement</td>
<td>-(Rn)</td>
<td>Register is decremented then used as a pointer.</td>
</tr>
<tr>
<td>110</td>
<td>Index</td>
<td>X(Rn)</td>
<td>Value X is added to (Rn) to produce address of the operand. Neither X nor (Rn) is modified.</td>
</tr>
</tbody>
</table>
TABLE 5.1b

Deferred or Indirect Addressing Modes of PDP-11

<table>
<thead>
<tr>
<th>Binary Code</th>
<th>Name</th>
<th>Assembler Syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Register</td>
<td>@Rn or (Rn)</td>
<td>Register contains the address of the operand.</td>
</tr>
<tr>
<td></td>
<td>Deferred</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>Autoincrement</td>
<td>@(Rn)+</td>
<td>Register is first used as a pointer to a word, then incremented by 2.</td>
</tr>
<tr>
<td></td>
<td>Deferred</td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>Autodecrement</td>
<td>@-(Rn)</td>
<td>Register is decremented by two and then used as a pointer to a word</td>
</tr>
<tr>
<td></td>
<td>Deferred</td>
<td></td>
<td>containing the address of the operand.</td>
</tr>
<tr>
<td>101</td>
<td>Index</td>
<td>@X(Rn)</td>
<td>Value X(stored in a word following the instruction) and (Rn) are added</td>
</tr>
<tr>
<td></td>
<td>Deferred</td>
<td></td>
<td>and the sum is used as a pointer to the word containing the address of</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the operand.</td>
</tr>
</tbody>
</table>
5.2 VALIDATION OF UNIT INSTRUCTION CONCEPT AND ESTIMATION OF TP

In this thesis, processor behavior has been modeled as an ordered sequence consisting of a memory request followed by some processing. A study of the PDP-11/20 instruction timing shown here exhibits a similar behavior. The access time starts when the Mp receives a request and ends when the data is received by the Pc.

Each PDP-11/20 instruction can be broken down into sequences of unit instructions. Figure 5.2 shows the instruction timing for the various addressing modes and instruction types. The PDP-11 Processor Handbook lists the total time for executing the various instructions. The access time is assumed to be 450 ns. Figure 5.2 is consistent with the handbook. Table 5.2 summarizes the contribution of the different cases to the effective processing time. Note that these figures take into account the delay due to the PDP-11 Unibus. Table 5.3 contains the instruction mix obtained by using Aygun's DAMET. Table 5.4 gives the effective processing times and their relative frequencies; the cumulative probability distribution function is plotted in Fig. 5.3. The access time of the memory was assumed to be 450 ns and the basic clock period of the PDP-11/20 processor was taken to be its nominal value of 140 ns. The average processing

†The author acknowledges B. Aygun's assistance in obtaining the instruction mix.
Single Operand Instructions & Double Operand Instructions with srcmode=0

Dstmode=0

fetch         execute         next instruction
560          700            560          560
  ta

Dstmode≠0

fetch         dest.          exec.         next instr.         fetch
560          700            210          280          560+490        560
  ta  Y  ta

Y :=

  ta                  dstmode = 1+2+4

  ta          700    ta                  dstmode = 3+5+6

  ta          700    ta          700    ta                  dstmode = 7

Figure 5.2 PDP-11/20 Instruction Timing
Double Operand Instructions with srcmode=0

Datmode=0  Srcmode=0

fetch  source  exec.  fetch

560  700  490  490  560  560

Y :=  ta  srcmode = 1+2+4

Datmode=0  Srcmode=0

fetch  source  destination  exec.  fetch

560  700  490  490  210  280  560+490  560

Fig. 5.2 (contd.)
Branch Instructions
Unsuccessful

Successful

Return Subroutine

Jump to Subroutine

Fig 5.2 (contd.)
TABLE 5.2

Effective Processing Time

<table>
<thead>
<tr>
<th>SINGLE OPERAND INSTRUCTIONS</th>
<th>tp in ns.</th>
<th>No. of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUBLE OPERAND WITH SRCMODE=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dstmode = 0</td>
<td>1820</td>
<td>1</td>
</tr>
<tr>
<td>dstmode = 1+2+4</td>
<td>910</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>550</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>840</td>
<td>1</td>
</tr>
<tr>
<td>dstmode = 3+5+6</td>
<td>910</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>550</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>840</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>700</td>
<td>1</td>
</tr>
<tr>
<td>dstmode = 7</td>
<td>910</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>550</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>840</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>700</td>
<td>2</td>
</tr>
</tbody>
</table>

DOUBLE OPERAND INSTRUCTIONS - SRCMODE=0

<table>
<thead>
<tr>
<th>srcmode=0 ∧ dstmode=0</th>
<th>tp in ns.</th>
<th>No. of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>common to all cases</td>
<td>1190</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1610</td>
<td>1</td>
</tr>
<tr>
<td>srcmode = 3+5+6</td>
<td>700</td>
<td>1</td>
</tr>
<tr>
<td>srcmode = 7</td>
<td>700</td>
<td>2</td>
</tr>
</tbody>
</table>
\begin{align*}
\text{srcmode} = 0 \land \text{dstmode} = 0 \\
\text{common to all cases} & \quad 1190 & 1 \\
& \quad 700 & 1 \\
& \quad 550 & 1 \\
& \quad 840 & 1 \\
\text{srcmode} = 3+5+6 & \quad 700 & 1 \\
\text{srcmode} = 7 & \quad 700 & 2 \\
\text{dstmode} = 3+5+6 & \quad 700 & 1 \\
\text{dstmode} = 7 & \quad 700 & 2 \\
\text{BRANCH INSTRUCTIONS} & \\
\text{successful} & \quad 2100 & 1 \\
\text{unsuccessful} & \quad 1260 & 1 \\
\text{RETURN SUBROUTINE} & \quad 1470 & 1 \\
& \quad 1850 & 1 \\
\text{JUMP SUBROUTINE} & \quad 1190 & 1 \\
& \quad 970 & 1 
\end{align*}
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single Operand Instructions</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dstmode=0</td>
<td>19 %</td>
<td></td>
</tr>
<tr>
<td>Dstmode=1+2+4</td>
<td>10 %</td>
<td></td>
</tr>
<tr>
<td>Dstmode=3+5+6</td>
<td>3 %</td>
<td></td>
</tr>
<tr>
<td>Dstmode=7</td>
<td>.1%</td>
<td></td>
</tr>
</tbody>
</table>

| **Double Operand Instructions with srcmode=0** |         |         |
| Dstmode=0            | 4.2 %   |         |
| Dstmode=1+2+4        | 5.75%   |         |
| Dstmode=3+5+6        | 3.55%   |         |
| Dstmode=7            | 0.0%    |         |

| **Double Operand : srcmode=0 . dstmode=0** |         |         |
| Srcmode=1+2+4        | 11.3%   |         |
| Srcmode=3+5+6        | 3 %     |         |
| Srcmode=7            | 0.1%    |         |

| **Double Operand : srcmode=0 . dstmode=0** |         |         |
| Srcmode=3+5+6        | 2 %     |         |
| Dstmode=3+5+6        | 2 %     |         |

| **Branch Instructions** |         |         |
| Successful             | 16 %    |         |
| Unsuccessful           | 11 %    |         |

| **Return & Jump to Subroutine** |         |         |
| Jump                      | 3 %     |         |
| Return                    | 3 %     |         |
Table 5.4

Relative Frequency Distribution of the Effective Processing Time

<table>
<thead>
<tr>
<th>Value in ns.</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>550</td>
<td>14.49 %</td>
</tr>
<tr>
<td>700</td>
<td>10.20 %</td>
</tr>
<tr>
<td>840</td>
<td>14.49 %</td>
</tr>
<tr>
<td>910</td>
<td>11.43 %</td>
</tr>
<tr>
<td>970</td>
<td>1.53 %</td>
</tr>
<tr>
<td>1050</td>
<td>1.53 %</td>
</tr>
<tr>
<td>1190</td>
<td>11.93 %</td>
</tr>
<tr>
<td>1260</td>
<td>5.61 %</td>
</tr>
<tr>
<td>1470</td>
<td>1.53 %</td>
</tr>
<tr>
<td>1610</td>
<td>7.35 %</td>
</tr>
<tr>
<td>1820</td>
<td>11.84 %</td>
</tr>
<tr>
<td>2100</td>
<td>8.16 %</td>
</tr>
</tbody>
</table>

Average Value = 1150 ns.
Figure 5.3  Cumulative probability distribution of the effective processing time of PDP-11/20
Chapter 5: Empirical Measurements, Parameter Estimation, Model Validation

5.2 Validation of Unit Instruction Concept, Estimation of $t_p$

The time obtained from this analysis is 1150 ns. The dotted curve in Fig. 5.3 is a shifted exponential distribution given by

$$
\text{Prob} \{ t_p \leq x \} = \begin{cases} 
1 - \exp \left[ -\frac{(x-450)}{700} \right] & \text{for } x \geq 450 \\
0 & \text{for } x < 450
\end{cases}
$$

5.3 MODEL VALIDATION VIA C.mmp PERFORMANCE EVALUATION

In this section the models are used to predict the performance of C.mmp and compared with actual measurements. Figure 5.4 shows the effect of D.map and the crosspoint switch on the parameters $t_p$, $t_a$ and $t_w$. The access time of the C.mmp core memory is 250 ns. However, a 200 ns nominal switch and memory control delay yields an effective $t_a$ of 450 ns. Each Mp is 8-way interleaved. Thus the rewrite time of 400 ns is overlapped with the next access if the next access is to one of the other 7 submodules of the Mp-module. Assuming random accessing within a module the average rewrite time experienced is only 50 ns. The effective average processing time is 1200 ns; the 50 ns delay associated with the relocation registers D.map is added to the basic value of 1150 ns.

The following experiment++ was conducted on the partial realization of C.mmp available to date. A program was loaded into one Mp-module and the three

++C.Pierson and W.Broadley were instrumental in the experimental set-up.
Legend:
1. Pc receives data from Mp
2. Pc finishes operating on the data
3. Pc has address of new data
4. Dmap computes physical address of data and puts it on the Pc-Mp bus.
5. Memory has received request at the crosspoint switch
6. Memory controller (part of crosspoint switch) selects the request to be served and sets up switch.
7. Data read from storage location
8. Data sent through switch
9. Memory recovers and starts serving next request (if any)

Figure 5.4 Unit instruction timing diagram for C.mmp
available processors executed the code individually, in pairs of two, and collectively. The number of memory cycles was measured. Table 5.5 presents the results of the experiment. The analytic results predicted by the geometric model of section 5.2 are depicted in Table 5.6; simulation results with processing time having the shifted exponential distribution of Fig. 5.3 are also listed. The analytic results are very close to the measured performance. However, for the 3x1 multiprocessor case the analytic model is about 10% higher than the measured value. This is due to the read-modify-write cycles which make the Mp service time greater than tc. The effect of these read-modify-write cycles is not crucial when the interference is not excessive. The simulation and analytic results show that the performance can be predicted by simple mathematical models with reasonable accuracy.
TABLE 5.5

Summary of Measurements on C.mmp

Number of Mp accesses per second

(Millions/sec)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.62015</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.61805</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.613657</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.14899</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.14672</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.14657</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.42466</td>
</tr>
</tbody>
</table>

0 - Pc is OFF


TABLE 5.6

Analytic Results

$E(tp) = 1200$ ns.

ta = 450 ns. tc = 500 ns.

<table>
<thead>
<tr>
<th>Number of Pcs</th>
<th>Number of Ms</th>
<th>Mp Access Rate (Millions/sec)</th>
<th>Simulation Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.60606</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1.14157</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1.56373</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2.39707</td>
<td>2.411</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>4.76009</td>
<td>4.751</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>9.48638</td>
<td>9.476</td>
</tr>
</tbody>
</table>
In the previous chapters several analytic models have been presented. Chapter 2 was devoted to systems in which the effective processing time is equal to the memory rewrite time. An important result observed was the absence of a law of diminishing returns. The performance of a multiprocessor system with n processors and n memories continues to rise at a constant rate as n increases. A simple exponential server model showed this rate to be 0.5; a constant processing time model predicted a slope of 0.586 for the average number of busy Mp's. The exponential server model gives the average number of busy Mp's as \( \frac{n+m}{n+m-1} \). An approximate result for constant processing times gives the average number of busy Mp's as \( \frac{1}{1-(1-1/i)^j} \), where \( i=\text{max}(n,m) \) and \( j=\min(n,m) \). An intuitively obvious conclusion limits the maximum number of active Pc's by \( \min(n,m) \). This maximum is reached if each processor accesses only one memory all the time. The model of section 2.6 analyzed the effect of skewing the processors’ access patterns. Thus, the maximum \( MP_{AR} \) is \( \min(m,n)/t_e \).

Chapter 3 contained several models for multiprocessor systems with \( tp>tw \). A new model for geometrically distributed processing time was developed. A different analytic approach was used to model systems with private caches for the processors. In general, since the Pc is slow, it takes fewer memory units
Chapter 6: Conclusions

for the performance to exhibit a saturation effect. This can be discerned by comparing Figs. 2.11 and 3.3a. In the absence of memory contention (which can now be possible even for m<n) the maximum $M_p/AR$ is $n/(ta*tp)$.

Chapter 4 discusses multiprocessor systems in which $tp<tw$. Since the processor is fast, performance improvement is obtained for $m>n$. If $m=n$, these systems do not yield significant improvement over systems with $tp=tw$. In general, adding an extra memory improves the performance more than adding an extra processor. The maximum average $M_p/AR$ is the minimum of $m/sc$ and $n/(ta*tp)$; the maximum is achieved if the processors do not interfere. Note that since the Pc is very fast, it can make a request to the memory that served it fast before the rewrite cycle is over. In this case, the Pc has to wait even though no other Pc is being serviced by the memory module.

Chapter 5 presented some empirical measurements of PDP-11 programs; C.mmp uses PDP-11's as the Pc's. It also illustrated how unit instructions can be extracted from the machine instructions. The measurement and the characterization of the processor's instruction timing was used to obtain the distribution of the effective instruction processing time, which was then used to evaluate analytically estimates of the unit execution rate of C.mmp. Contrary to common intuition, the unit instruction concept yields a larger effective processing time for register-register operations, since the processing time is a measure of the interval between two memory accesses. The average processing time was found to be 1150 ns. for the PDP-11/20. The average processing times for the
Chapter 6 : Conclusions

PDP-11/40 and 11/45 are approximately 625 and 400 nanoseconds. A PDP-11/20 instruction can require from 1 to 7 accesses to memory, but the average PDP-11/20 instruction makes approximately 2 accesses to memory; it comprises two unit instructions.

6.1 APPLICATIONS

The models developed in this thesis should give computer system designers considerable insight into some of the design issues. If $t_p > t_w$, the performance (as indicated by the $M_p/R$) saturates for some value of $m$ that is greater than $n$. If $t_p < t_w$, however, it may saturate in some cases for $m < n$. Most system designers tend to use optimality of cost/performance as an over-riding factor. Another important issue is the extendability or the effect of changing the system parameters [BhatS72]. Performance requirements often change and the initial design should be modifiable to meet the new performance desired. A good extendable design should therefore have a certain amount of unutilized capacity. The system should be designed to have a performance slightly greater than that which maximizes performance/cost.

In a multiprocessor system, the alternatives for change include increasing the number of Pcs or Mp's, as well as replacing the Mp's or Pc's with faster versions. The adding of Pcs and Mp's is a less viable alternative due to the
extent of the engineering effort involved. In some cases, the entire central
crosspoint switch may have to be either redesigned or rebuilt. However,
replacement with faster upward compatible processors or faster memories is
easier. It is an undisputed fact that increasing the speed of the processor or
memory by a factor of \(k\) will not enhance the performance by the same factor. The
analytic models suggest that increasing the processor speed causes the
saturation point to shift to a larger \(m\). If \(t_P < t_W\) a well-designed system is
likely to have \(m < n\). Since the memory is the limiting factor, the best choice for
enhancing the performance is the use of faster memories. However, if the
original design used more memory modules than needed to saturate the
performance, faster processors will also yield some improvement. If \(t_P > t_W\)
substitution of faster processors effects an increase in the access rate. To
allow for future substitution of faster processors it is desirable to choose
\(m > n\). Thus, a rule of thumb indicates that a good design choice is \(m > n\), with a
not too large a mismatch in \(P_c\) and \(M_p\) speeds. As described in chapter 5, C.mmp
is a 16x16 multiprocessor system with \(t_P = 1200\) ns, \(t_a = 450\) ns and \(t_W = 50\) ns. The
\(M_p/IR\) expected is 9.476 million/sec. If the PDP-11/20 processor is replaced by a
faster \(P_c\) (11/45) with a typical processing time of 450 ns the \(M_p/IR\) increases to
15.6 million/sec. However, since the system is very much processor speed
limited, a 150 ns cache for each \(P_c\) increases the \(M_p/IR\) only by 14\% and 20\% for
hit ratios of 0.5 and 0.7 respectively.
6.1.1 An Illustrative Example

In order to illustrate the use of the models developed in this thesis, consider an 8x8 multiprocessor system with \( tp=tw \). This configuration is still memory limited, as seen from Fig. 2.10. Let the memory cycle time be 950 ns; access time = 400 ns. Assume a switch delay of 200 ns and a 50 ns delay in the relocation hardware. The processor has a typical processing time of 450 ns. Hence, the effective processing time is 500 ns; effective access time is 600 ns; and the effective cycle time is 1100 ns (assuming that 50 ns of the switch delay is overlapped with the rewrite cycle).

The discrete Markov chain model of Chapter 2 gives the average number of busy Mp's to be 4.9471; the \( Mp/IR \) is 4.4974 million/sec. Let us evaluate the effect of replacing the original Pc with either a 300 ns or a 150 ns processor. Note that the effective processing time is 350 ns and 200 ns. The corresponding percentage increase in the \( Mp/IR \), as computed by the models of chapter 4, is 4.33% and 9.05%. This is not surprising since the original system was memory speed limited.

Instead of changing the Pc's, the Mp's could be changed. If the new Mp has \( ta = 250 \) ns and \( tw = 300 \) ns, the effective \( ta \) and \( tw \) are 450 ns and 250 ns respectively. The new cycle time is 700 ns. Since \( tp > tw \) the models of chapter 3 can be used. The new Mp access rate is 44.39% higher.

A third alternative is the use of a 150ns access time cache. The value of
Chapter 6: Conclusions

6.1 Applications

\( \alpha \), the probability of finding the data in the cache, depends on the size of the cache. Consider three sizes that result in \( \alpha = 0.5 \), \( \alpha = 0.7 \) and \( \alpha = 0.8 \). The cache model of chapter 3 predicts performance enhancement of 67\%, 107\% and 130\%.

The above performance data coupled with cost information can be used to select a profitable parameter change. When the number of design alternatives is large, simple analytic models help to determine a judicious choice.

6.2 PROPOSALS FOR FUTURE WORK

This thesis is not a panacea for multiprocessor system designers. An attempt has been made to develop some simple basic tools. Anyone working with systems is aware of their high degree of complexity and is likely to be shocked when he sees the simplicity of the models suggested. He may react negatively when he notices how much of the real system has been left out and how restrictive the assumptions are that have been made in the analysis. This skepticism is not entirely justified; simple analytic models often exhibit overall behavior similar to the complex system modeled.

One of the main assumptions made in this thesis is the independence of successive memory requests. In most real systems, due to program locality, there is some serial correlation between requests made by a PC. Thus, if a PC accesses
Chapter 6: Conclusions

6.2 Proposals for Future Research

Mp[j] it continues to do so for some amount of time. A modeling technique suggested is dividing the system activities into various phases. The number of Mp's accessed in each phase is different. Jackson's general exponential server model[Jack.63] indicates that the stationary state probabilities depend only on the average frequency of visits to the various servers. Therefore the effect of the serial correlation of requests is not seen. However, for real systems which are not exponential, some degradation may be observed. We also assume p_{1j}/m. As indicated in section 2.6, this is not the most desirable access pattern. The effect of skewing the access patterns is to increase the MP/AR. Given the above two opposing effects, p_{1j}/m serves as a good parameter value for comparison at a high level.

Since no a priori empirical evidence is available, a large portion of future research activity should involve the measurement of real systems. This will bring to light the seriousness of the assumptions and establish a proper framework and area of applicability of analytic models. Measurement of dynamic program behavior should be stressed. It should be remembered that the output of the models can only be as good as the input. System analysts should not neglect the parameter estimation phase of performance prediction.

Future analytic studies should attempt to differentiate between instruction and data references. At a higher level, memory interference models can be used as a part of an overall hierarchical model of the computer system at the program level. Empirical studies should be conducted in order to obtain good concise
characterization of component and subsystem behavior. Such information can be
used to drive more efficient simulations of complex systems.
REFERENCES


Appendix A-1  Listing of Program for the algorithm shown in Fig. 2.5.

Appendix A-2  Listing of Program for the algorithm shown in Fig. 2.8.

Appendix A-3  Approximate Markov Chain Model for tp=tw.

Appendix A-4  Approximate Model for Arbitrary $R_j$, $tp=tw$, max.

Appendix A-5  Approximate Markov Chain Model for tp=tw+tc.

Appendix A-6  Approximate Markov Chain Model for tp>tw.

Appendix A-7  Cache Model for tp>tw.
APPENDIX A-1

LISTING OF PROGRAM FOR THE ALGORITHM SHOWN IN FIG. 2.5

DISCRETE MARKOV CHAIN TP=T4 -- STATE(NO. OF STATES, NO. OF PC)

INTEGER STACK(16,16),STATE(25,16),A(16),FIRST(16)
1, IDONE(25), PTR(16), DONE(16), NWAYS(16), BOUND(17)
DIMENSION TRANS(25,25), B(25), Z(25)

COMON TRANS,B,Z,NPARTS

INTEGER SUM,T,X,Q

TYPE 1973

FORMAT(1X,'NUMBER OF PROCESSORS',/)

ACCEPT 1974,NPC

TYPE 1974

FORMAT(I)

ACCEPT 1975

FORMAT(1X,'NUMBER OF MEMORIES',/)

MINPM=MIN0(NPC,NMP)

M=1

PTR(NPC-1)=1

GO TO 31

CONTINUE

IF (NPC.GT.1) PTR(NPC-1)=NPM+1

DO 21 I=2,M

A(I)=1

30 SUM=0

DO 22 I=2,M

SUM=SUM+A(I)

A(I)=NPC-SUM

WRITE(15,2000),(A(I),I=1,M)

FORMAT(1X,16(I3.1X))

C

2000

NPARTS=NPARTS+1

DO 17 K=1,M

STATE(NPARTS,K)=A(K)

C

T=2

CONTINUE

IF(T.GT.M) GO TO 120

X=A(T)-A(T)

IF(X.GT.1) GO TO 100

T=T+1
Appendix A-1
Listing of Program for the Algorithm shown in Fig. 25

GO TO 60
       CONTINUE
       ITMP=A(T)
       DO 101 I=2,T
       A(I)=ITMP+1
       GO TO 30
       100

       120
       M=M+1
       BOUND(M)=NPARTS
       WRITE(15,2001),NPARTS
       2001
       FORMAT(/,1X,'***********',13,'***********',/) 
       IF (M.LE.MINPM) GO TO 20
       WRITE(15,111),NPARTS
       111
       FORMAT(/,1X,'NUMBER OF PARTITIONS=',I)
       WRITE(15,3113)
       3113
       FORMAT(/)

       C

       C

       C

       C

       C

       INITIALIZE THE STACK, NWAYS, FIRST
       DO 2100 I=1,NPC-1
       STACK(I,1)=1
       NWAYS(I)=1
       2100
       FIRST(I)=2
       FIRST(NPC-1)=1
       C

       C

       BEGIN WALK THROUGH TREE
       C
       L=NPC
       CONTINUE
       IF(DONE(L).EQ.1) GO TO 25
       C
       200
       K=L-1
       J=FIRST(K)
       C
       DO 10 I=J,NMP
       IF(STACK(K,I).NE.STACK(K,J)) GO TO 15
       10
       CONTINUE
       C
       DONE(L)=1
       FIRST(K)=1
       I=I+1
       C
       15
       NWAYS(L)=L-J
       C
       DO 300 KK=1,NMP
Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

```
300       STACK(L, KK) = STACK(K, KK)  
           STACK(L, J) = STACK(K, J) + 1
C
         IF(DONE(L).NE.1) FIRST(K) = 1
C
         IF(L.EQ.NPC) GO TO 1000
C
C SET PTR TO ORIGINAL STATE AT LEVEL L
C
         IF(PTR(L).NE.0) IDONE(PTR(L)) = 1
         K = NFC - L
         K1 = BOUND(K) + 1
         K2 = BOUND(K+1)
C
         DO 35 JJ = K1, K2
C
         DO 32 KK = K, 1, -1
C
           IF(STATE(KK, JJ).EQ.STACK(L, JJ) + 1) GO TO 34
           CONTINUE
C
           KK = 0
           GO TO 36
34
         K1 = KK
C
35         CONTINUE
C
36         PTR(L) = KK
C
C
         L = L + 1
         GO TO 200
C
C
25         DONE(L) = 0
         L = L - 1
         IF(L.EQ.1) GO TO 7777
C
         GO TO 1
C
C FIND TERMINAL STATE
C
1000       CONTINUE
```
Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

DO 1100 K=1,NMP
   IF(STACK(NPC,K).EQ.0)GO TO 1150
1100    CONTINUE
K=NMP+1
C
1150    K=K-1
   K1=BOUND(K)+1
   K2=BOUND(K+1)
C
   DO 135 J=K,1,-1
   DO 132 K1,K2
      IF(STATE(KK,JJ).EQ.STACK(NPC,JJ))GO TO 134
      CONTINUE
      TYPE 9876
      FORMAT(1X,'CANNOT FIND TERMINAL STATE !!!!!!')
      K1=KK
      CONTINUE
      C
      UPDATE TRANSITION MATRIX
      TEMP=1
      DO 150 I=NPC-1,1,-1
         II=PTR(I)
         TEMP=TEMP*WAYS(I+1)
         IF(II.EQ.0)GO TO 150
         IF(IDONE(II).NE.1)TRANS(KK,II)=TRANS(KK,II)+TEMP
      CONTINUE
      C
      IF(NPC.LE.NMP)TRANS(KK,NPARTS)=TRANS(KK,NPARTS)+TEMP*NMP
      GO TO 1
      C
      TRANSITION MATRIX HAS BEEN GENERATED
      C
      7777    CONTINUE
      C
      DO 5544 I=1,NPARTS
      C5544    WRITE(15,4455),(TRANS(I,J),J=1,NPARTS)
      C4455    FORMAT(1X,25(F8.4,1X))
      C
      DO 7250 I=1,MINPM
         Y=NMP*X
         CONTINUE
      C
Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

Page 145

DO 7250 J=BOUND(I)+1,BOUND(I+1)
DO 7100 K=1,NPARTS
   TRANS(K,J)=TRANS(K,J)/Y
   CONTINUE
C
DO 5544 I=1,NPARTS
   WRITE(15,4455),(TRANS(I,J),J=1,NPARTS)
   FORMAT(1X,25(F8.6,1X))
C
DO 8000 I=1,NPARTS
   TRANS(I,I)=TRANS(I,I)-1.0
   TRANS(NPARTS,I)=1.0
C
   B(NPARTS)=1
   CALL GAUSS
   DO 8500 I=1,MINPM
      TMP=0
      DO 8250 J=BOUND(I)+1,BOUND(I+1)
         TMP=TMP+Z(J)
      C
      OCC=OCC+TMP*1
      WRITE(15,8750),OCC
      FORMAT(//,1X,'EXPECTED VALUE OF NO. OF BUSY MP= ' , 1,F10.6)
C
STOP
END
C
SUBROUTINE GAUSS
THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
A·X=B
C
DIMENSION A(25,25),B(25),X(25)
COMMON A,B,X,N
INTEGER S
C
S=N
5 CONTINUE
   IF(S-1) 50,50,10
10 CONTINUE
   IF(S.GT.2)GO TO 105
   D=A(1,1)·A(2,2)-A(1,2)·A(2,1)
Appendix A-1
Listing of Program for the Algorithm shown in Fig. 2.5

```
IF(ABS(D).GT.0.0005)GO TO 105
TYPE 25
GO TO 100

105   DO 20 I=1,S
      M=S-I+1
      IF(ABS(A(M,S)).GT.0.0005)GO TO 30
      CONTINUE

20     TYPE 25

25    FORMAT(IX,'THE COEFFICIENT MATRIX IS SINGULAR',/)
      GO TO 100

30     CONTINUE
      IF(M.EQ.S)GO TO 40

40     T=B(S)
      B(S)=B(M)
      B(M)=T
      DO 35 J=1,S
      T=A(S,J)
      A(S,J)=A(M,J)
      A(M,J)=T
      CONTINUE

35     CONTINUE

45     CONTINUE
      DO 45 I=1,S-1
      K=S-1
      IF(ABS(A(S,S)).GT.0.0005)GO TO 42
      GO TO 100

42     B(K)=B(K)-B(S)*A(K,S)/A(S,S)
      DO 45 J=1,S-1
      A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
      CONTINUE
      S=S-1
      GO TO 5

50     CONTINUE
      DO 70 I=1,N
      SUM=B(I)
      DO 60 J=1,I-1
      SUM=SUM-A(I,J)*X(J)
      CONTINUE

60     CONTINUE
      IF(ABS(A(I,I)).GT.0.0005)GO TO 61
      GO TO 100

61     X(I)=SUM/A(I,I)

70     CONTINUE

100   RETURN
      CONTINUE
      END
```
APPENDIX A-2

LISTING OF PROGRAM FOR THE ALGORITHM SHOWN IN FIG. 2.8

************************************************************************************************

THIS PROGRAM SOLVES THE DISCRETE MARKOV CHAIN FOR TP-TW USING SINGLE LEVEL TRANSITIONS
LONG VERSION

************************************************************************************************

INTEGER STATE(186,16),BOUND(17),B1(17),B2(17)
INTEGER S1(116,16),S2(146,16)
DIMENSION TRANS(186,186),B(186),Z(186),X12(146,116),X23(186,146)
DIMENSION C(186)
COMMON NPARTS,TRANS,B,Z
INTEGER T,Q
INTEGER A(16),BD(17)

TYPE 1973
1973 FORMAT(IX,'NUMBER OF PROCESSORS',/)
ACCEPT 1974,NPC
1974 FORMAT(I)
TYPE 1975
1975 FORMAT(IX,'NUMBER OF MEMORIES',/)
ACCEPT 1974,NMP
WRITE(15,1949),NPC,NMP
1949 FORMAT(IX,'NUMBER OF PROCESSORS =',12,/,1X,'NUMBER OF MEMORIES =',12,/
MINPM=MINB(NPC,NMP)
LEVEL=NPC

C
IPRT=1
GO TO 4000

C
PARTITION NPC INTO NMP PARTS
C
FINAL STATE VECTOR --- STATE
C
421 NPARTS=NNPART
4421 DO 4421 I=1,M
BOUND(I)=BD(I)

C
IPRT=2
LEVEL=NPC-1
GO TO 4000

C
PARTITION NPC-1
Listing of Program for the Algorithm shown in Fig. 2.8

PARTIAL STATE VECTOR --- S2

DO 4422 I=1,M

GENERATE NUMBER OF WAYS OF GOING FROM S2 TO STATE

LSRC=MIN0(NPC-1,NMP)
DO 1 I=1,NPARTS
DO 1 J=1,N2

DO 40 K=1,LSRC
I1=B2(K)+1
I2=B2(K+1)
DO 40 L=I1,I2
K1=BOUND(K)+1
K2=BOUND(K+2)
IF(M\textsuperscript{NMP}.LT.K+1)K2=BOUND(K+1)
DO 32 KK=K1,K2
M=0
JJ=MIN0(MINPM,K+1)
DO 20 I=1,JJ
IF(S2(L,I).EQ.STATE(KK,I))GO TO 20
IF(S2(L,I).NE.STATE(KKI)-1)GO TO 32
M=M+1
II=I
20 CONTINUE
IF(M.NE.1)GO TO 32
DO 25 I=II,NMP
IF(S2(L,I).NE.S2(L,II))GO TO 27
25 CONTINUE
I=I+1
27 X23(KK,L)=I-11
32 CONTINUE
40 CONTINUE

UPDATE TRANSITION MATRIX --- TRANS

DO 100 L=1,N2

DO 35 JJ=K,1,-1
Listing of Program for the Algorithm shown in Fig. 2.8

C
DO 5632 KK=K1,K2
IF (STATE(KK,JJ).EQ.S2(J,J)+1) GO TO 34
5632
   CONTINUE
C
   KK=0
GO TO 36
34
   K1=KK
C
   CONTINUE
35
C
   CONTINUE
36
   CONTINUE
IF (KK.EQ.0) GO TO 100
DO 50 I=1,NP1RT
50
   TRANS(I,KK)=TRANS(I,KK)+X23(I,L)
100
   CONTINUE
C
   DECREMENT LEVEL
C
   LEVEL=NP1RT-2
200
   CONTINUE
   IF (LEVEL.EQ.0) GO TO 300
C
   PARTITION LEVEL
C
   PARTIAL STATE VECTOR --- S1
C
   IPR1T=3
GO TO 4000
423
   N1=NP1RT
DO 4423 I=1,M
4423
   B1(I)=BD(I)
C
   COMPUTE NWAYS FOR GOING FROM S1 TO S2
C
   LSRC=MIN0(LEVEL,NP1RT)
C
   DO 341 I=1,N2
   DO 341 J=1,N1
341
   X12(I,J)=0
   DO 5740 K=1,LSRC
      I1=B1(K)+1
      I2=B1(K+1)
      DO 5740 L=I1,I2
         K1=B2(K)+1
         K2=B2(K+2)
         IF (MINPM.LT.K+1) K2=B2(K+1)
   5740
Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

DO 5732 KK=K1,K2
M=0
JJ=MIN0(MINPM,K+1)
DO 577 I=1,JJ
IF(S1(L,I).EQ.S2(KK,I))GO TO 577
IF(S1(L,I).NE.S2(KK,I)-1)GO TO 5732
M=M+1
II=1
577 CONTINUE
IF(M.NE.1)GO TO 5732
DO 5725 I=II,NMP
IF(S1(L,I).NE.S1(L,II))GO TO 979
CONTINUE
1255 CONTINUE
IC=1
GO TO 125

K=NPC-LEVEL
IF(K.GT.MINPM)GO TO 5800
K1=BOUND(K)+1
K2=BOUND(K+1)

DO 5800 L=1,N1
DO 5835 JJ=K1,K2
DO 5832 KK=K1,K2
IF(STATE(KK,JJ).EQ.S1(L,JJ)+1)GO TO 5834
5832 CONTINUE
KK=0
GO TO 5836
5834 K1=KK
CONTINUE
5835 CONTINUE
5836 CONTINUE
Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

IF (KK.EQ.0) GO TO 5800
DO 5850 I=1,NPARTS

5850 TRANS(I,KK)=TRANS(I,KK)+X23(I,L)
5800 CONTINUE

C
C DECREMENT LEVEL
C
LEVEL=LEVEL-1
IF (LEVEL.EQ.0) GO TO 300

C
C PARTITION LEVEL
C PARTIAL STATE VECTOR --- S2
C
IPRT=4
GO TO 4000

424 DO 4424 I=1,M
N2=NNPART

4424 B2(I)=BD(I)

C
C COMPUTE NWAYS FROM S2 TO S1
C
LSRC=MIN0(LEVEL,NMP)

C
DO 591 I=1,N1
DO 591 J=1,N2

591 X12(I,J)=0
DO 5940 K=1,LSRC
I1=B2(K)+1
I2=B2(K+1)
DO 5940 L=I1,I2
K1=B1(K)+1
K2=B1(K+2)
IF (MINPM.LT.K+1) K2=B1(K+1)
DO 5932 KK=K1,K2
M=0
JJ=MIN0(MINPM,K+1)
DO 5920 I=1,JJ
IF (S2(L,I).EQ.S1(KK,I)) GO TO 5920
IF (S2(L,I).NE.S1(KK,I)-1) GO TO 5932
M=M+1
I=1

5920 CONTINUE
IF (M.NE.1) GO TO 5932
DO 5925 I=1,NMP
IF (S2(L,I).NE.S2(L,I)) GO TO 5927

5925 CONTINUE
Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

I=I+1
5927 X12(KK,L)=I-I1
5932 CONTINUE
5940 CONTINUE
C
C MATRIX MULTIPLICATION GIVES WAYS FROM S2 TO STATE
C
IC=2
GO TO 125
1260 CONTINUE
C
C UPDATE TRANS
C
K=NPC-LEVEL
IF(K.GT.MINPM)GO TO 6300
K1=BOUND(K)+1
K2=BOUND(K+1)
C
DO 6300 L=1,N2
C
DO 6335 JJ=K,1,-1
C
DO 6332 KK=K1,K2
IF(STATE(KK,JJ).EQ.S2(L, JJ)+1)GO TO 6334
6332 CONTINUE
C
KK=0
GO TO 6336
6334 K1=KK
C
6335 CONTINUE
C
6336 CONTINUE
IF(KK.EQ.0)GO TO 6300
DO 6350 I=1,NPARTS
6350 TRANS(I,KK)=TRANS(I,KK)+X23(I,L)
6380 CONTINUE
C
C DECREMENT LEVEL
C
LEVEL=LEVEL-1
GO TO 200
300 CONTINUE
IF(NPC.GT.NMP)GO TO 400
DO 350 I=1,NPARTS
350 TRANS(I,NPARTS)=TRANS(I,NPARTS)+0.0*NMP*X23(I,1)
CONTINUE

TRANSITION MATRIX HAS BEEN GENERATED

CONTINUE

DO 5544 I=1,NPARTS
WRITE (15,4455), (TRANS(I,J), J=1,NPARTS)
C4455 FORMAT (1X,25(F8.4,1X))
C
DO 7250 I=1,MINPM
Y=(1.0**NMP)**I
DO 7250 J=BOUND(I)+1,BOUND(I+1)
DO 7100 K=1,NPARTS
TRANS(K,J)=TRANS(K,J)/Y
7100 CONTINUE
7250 CONTINUE

DO 5544 I=1,NPARTS
WRITE (15,4455), (TRANS(I,J), J=1,NPARTS)
C4455 FORMAT (1X,25(F8.6,1X))
C
DO 8000 I=1,NPARTS
TRANS(I,I)=TRANS(I,I)-1.0
8000 TRANS(NPARTS,I)=1.0
C
B(NPARTS)=1
C
CALL GAUSS
C
DO 8500 I=1,MINPM
C
TMP=0
DO 8250 J=BOUND(I)+1,BOUND(I+1)
8250 TMP=TMP+Z(J)
WRITE (15,9321), I, TMP
9321 FORMAT (1X,'PROB(NO. OF BUSY MEMS=',12,')=',1X,F10.6)
C
8500 OCC=OCC+TMP
C
WRITE (15,8750), OCC
8750 FORMAT (1X,'EXPECTED VALUE OF NO. OF BUSY MP=',1,F10.6)
C
STOP
C
C MATRIX MULTIPLICATION IN-LINE SUBROUTINE
Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

125    CONTINUE
       N=NPARTS
       NM=N2
       M=N1
       IF(IC.EQ.1)GO TO 126
       NM=N1
       M=N2

126    CONTINUE
       DO 511 I=1,N
       DO 511 J=1,M
       C(J)=0
       DO 511 K=1,NM
          C(J)=C(J)+X23(I,K)*X12(K,J)
       DO 511 J=1,M
       X23(I,J)=C(J)
       IF(IC.EQ.1)GO TO 1255
       GO TO 1260
       STOP

C     IN-LINE SUBROUTINE FOR GENERATING PARTITIONS
     STATE VECTORS

4000    MINL=MIN0(LEVEL,NMP)
        NNPART=0
        SUM=0
        M=1
        GO TO 31

220    CONTINUE
       DO 21 I=2,M
       A(I)=1
       30    SUM=0
       DO 22 I=2,M
       SUM=SUM+A(I)
       31    A(I)=LEVEL-SUM
       WRITE(15,2000) (A(I),I=1,M)
       2000    FORMAT(1X,16(I3,1X))
        NNPART=NNPART+1
        GO TO (171,172,173,172),IPRT

171    CONTINUE
       DO 1711 K=1,M
       1711    STATE(NNPART,K)=A(K)
Listing of Program for the Algorithm shown in Fig. 2.8

```fortran
IF(NMP.LE.M)GO TO 1700
DO 181 K=M+1,NMP
  181 STATE(NNPART,K)=0
172 GO TO 1700
   CONTINUE
DO 1722 K=1,M
  1722 S2(NNPART,K)=A(K)
IF(NMP.LE.M)GO TO 1700
DO 182 K=M+1,NMP
  182 S2(NNPART,K)=0
173 GO TO 1700
   CONTINUE
DO 1733 K=1,M
  1733 S1(NNPART,K)=A(K)
IF(NMP.LE.M)GO TO 1700
DO 183 K=M+1,NMP
  183 S1(NNPART,K)=0
1700 T=2
60 CONTINUE
IF(T.GT.M)GO TO 1220
IIX=A(I1)-A(T)
IF(IIX.GT.1)GO TO 1000
  T=T+1
GO TO 60
1000 ITMP=A(T)
DO 101 I=2,T
  101 A(I)=ITMP+1
GO TO 30
1220 BD(M)=NNPART
C WRITE(15,2201),NNPART
  2201 FORMAT(/,IX,'**********',I3,'**********',//)
IF(M.LE.MINL)GO TO 220
GO TO (421,422,423,424),IPRT
C
C END

C SUBROUTINE GAUSS
C THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
  \[ A \times X = B \]
C
DIMENSION A(186,186),B(186),X(186)
COMMON N,A,B,X
INTEGER S
```
Listing of Program for the Algorithm shown in Fig. 2.8

C
5 S=N
  CONTINUE
  IF(S-I) 50, 50, 10
10 CONTINUE
  IF(S.GT.2) GO TO 105
  D=A(1,1)×A(2,2)-A(1,2)×A(2,1)
  IF(ABS(D).GT.0.0005) GO TO 105
  TYPE 25
  GO TO 100
105 DO 20 I=1,S
  M=S-I+1
  IF(ABS(A(M,S)).GT.0.0005) GO TO 30
20 CONTINUE
  TYPE 25
25 FORMAT(IX, 'THE COEFFICIENT MATRIX IS SINGULAR', I)
  GO TO 100
30 CONTINUE
  IF(M.EQ.S) GO TO 40
  T=B(S)
  B(S)=B(M)
  B(M)=T
  DO 35 J=1,S
    T=A(S,J)
    A(S,J)=A(M,J)
    A(M,J)=T
35 CONTINUE
40 CONTINUE
  DO 45 I=1,S-1
    K=S-I
    IF(ABS(A(S,S)).GT.0.0005) GO TO 42
  GO TO 100
42 B(K)=B(K)-B(S)×A(K,S)/A(S,S)
  DO 45 J=1,S-1
    A(K,J)=A(K,J)-A(K,S)×A(S,J)/A(S,S)
45 CONTINUE
  S=S-1
  GO TO 5
50 CONTINUE
  DO 78 I=1,N
    SUM=B(I)
  DO 68 J=1,I-1
    SUM=SUM-A(I,J)×X(J)
68 CONTINUE
  IF(ABS(A(I,I)).GT.0.0005) GO TO 61
  GO TO 100
Appendix A-2
Listing of Program for the Algorithm shown in Fig. 2.8

61       X(I)=SUM/A(I,I)
70       CONTINUE
100      CONTINUE
       RETURN
       END
APPENDIX A-3

APPROXIMATE MARKOV CHAIN MODEL FOR TP-TW

DIMENSION S(17,17), CM(16,16), TRANS(17,17)
INTEGER COMB(16,16)
DIMENSION Z(17), B(17)
INTEGER D
COMMON TRANS, B, Z, MINPM

TYPE 1973
ACCEPT 1974, N
TYPE 1975
ACCEPT 1974, M
X = 1.0 * M
MINPM = MIN0(N, M)

DO 300 I = 1, 17
   S(I, 1) = 0
300   S(1, 1) = 0
   S(1, 1) = 1

DO 400 I = 1, 16
   DO 400 J = 1, I
   400   S(I + 1, J + 1) = 1.0 * J * S(I, J + 1) + S(I, J)

DO 500 I = 1, 16
   CM(I, 1) = 1
   DO 500 J = 2, I
   500   CM(I, J) = CM(I, J - 1) * 1.0 * (I - J + 1)

COMB(1, 1) = 1
DO 20 K = 2, 16
   COMB(K, 1) = K
   L = MIN0(K, 16)
   DO 10 I = 2, L
10   COMB(K, I) = COMB(K, I - 1) * (K - I + 1) / I
Appendix A-3
Approximate Markov Chain Model for \(tp=tw\)

20 CONTINUE

DO 1000 I=1,MINPM
   XI=1.0/XM
   DO 700 J=0,I
   J IS THE NUMBER OF ACTIVE PC'S THAT MAKE A REQUEST TO THE MP'S
   XPROB IS THE PROB THAT N-I+J PC MAKE A REQUEST TO THE MP'S
   IF (J.EQ.0) XPROB=(1.0-XI)*XI
   XMULT=(1.0-XI)*XI*(I-J)
   IF (I.EQ.J) XMULT=1.0
   IF (J.NE.0) XPROB=1.0*COMB(I,J)*XI*XI*(I-J)*XMULT
   NN=N-I+J
   K1=MIND(NN,1)
   L1=1
   IF (K1.EQ.0) L1=0
   DO 700 L1=1,K1
   TEMP1=1
   IF (L1.EQ.0) GO TO 595
   X1=(1.0*X1)*NN
   TEMP1=CM(1,L1)*S(NN+1,L1+1)/XI
   NBUSY=NO. OF BUSY MP'S DURING NEXT CYCLE
   K2=MIND(1-I,J-1)
   L2=1
   IF (K2.EQ.0) L2=0
   DO 700 L2=1,K2
   TEMP2=1
   NBUSY=L1+L2
   IF (L2.EQ.0) GO TO 695
   X2=(1.0*X2)*(M-I)*(I-J)
   TEMP2=CM(M-I,L2)*S(1-J+1,L2+1)/X2
   695 TRANS(NBUSY,1)=TRANS(NBUSY,1)+XPROB*TEMP1*TEMP2
   700 CONTINUE
   1000 CONTINUE
   DO 8000 I=1,MINPM
      TRANS(I,1)=TRANS(I,1)-1.0
      8000 TRANS(MINPM,1)=1.0
      B(MINPM)=1
      CALL GAUSS
Appendix A-3
Approximate Markov Chain Model for $t_p$-tw

C

DO 2000 I=1,MINPM
UER=UER+1.0*Z(I)*I
2000 CONTINUE
C
TYPE 1,UER
C
FORMAT(1X,F10.6)
C
STOP
END

C
SUBROUTINE GAUSS
C
THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
A*X=B
C
DIMENSION A(17,17),B(17),X(17)
COMMON A,B,X,N
INTEGER S

S=N
CONTINUE
IF(S-1) 50,50,10
C
IF(S.GT.2)GO TO 105
D=A(1,1)*A(2,2)-A(1,2)*A(2,1)
IF(ABS(D).GT.0.0005)GO TO 105
TYPE 25
GO TO 100
105
DO 20 I=1,S
M=S-I+1
IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20 CONTINUE
TYPE 25
25 FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR',/)
GO TO 100
30 CONTINUE
IF(M.EQ.S)GO TO 40
T=B(S)
B(S)=B(M)
B(M)=T
DO 35 J=1,S
T=A(S,J)
A(S,J)=A(M,J)
A(M,J)=T
35 CONTINUE
Appendix A-3
Approximate Markov Chain Model for t_p-t_w

40 CONTINUE
   DO 45 I=1,S-1
      K=S-1
      IF(ABS(A(S,S)).GT.8.8805)GO TO 42
      GO TO 100
42   B(K)=B(K)-B(S)*A(K,S)/A(S,S)
   DO 45 J=1,S-1
      A(K,J)=A(K,J)-A(K,S)*A(S,J)/A(S,S)
45 CONTINUE
   S=S-1
   GO TO 5
50 CONTINUE
   DO 70 I=1,N
      SUM=B(I)
   DO 60 J=1,I-1
      SUM=SUM+A(I,J)*X(J)
60 CONTINUE
   IF(ABS(A(I,I)).GT.8.8805)GO TO 61
   GO TO 100
61 X(I)=SUM/A(I,I)
70 CONTINUE
100 RETURN
APPENDIX A-4

*/

APPROXIMATE MODEL FOR ARBITRARY P(I,J), TP=TW, M>N.

\[ P(I,J) = \begin{cases} \alpha & \text{for } I=J \\ b & \text{otherwise} \end{cases} \]

DIMENSION UER(21), ALPHA(21), PROB(16,16)
DIMENSION QPR(16,16)
DIMENSION RATE(16), Y(16,16)
DATA ALPHA/0.0, .05, .15, .25, .35, .45, .5, .55, .6, 1.65, 7.75, 8.85, .95, 1.0/
TYPE 1973

1973 FORMAT(IX,'NUMBER OF PROCESSORS',/)
1974 ACCEPT 1974,NPC
1975 TYPE 1975
1975 FORMAT(IX,'NUMBER OF MEMORIES',/)
1975 ACCEPT 1974,NMP

\[ X = N \min(NMP,NPC) \]
\[ XN = 1.0 \times \text{NPC} \]

COMPUTE UER FOR VARIOUS VALUES OF ALPHA

DO 9999 IJK=1,21
   A=ALPHA(IJK)
   B=(1.0-A)/(XM-1)
   DO 11 Ul.NPC
      DO 18 J=1,NMP
         PROB(I,J)=B
         PROB(I,I)=A
   CONTINUE
11 CONTINUE

COMPUTE QUEUEING FREQUENCIES BASED ON ACCESS FREQUENCIES.

DO 78 N1,NPC
   DO 78 J=1,NMP
      Y(I,J)=1.0
   CONTINUE
78 CONTINUE

\[ Y(I,J) = Y(I,J) \times (1.0 - \text{PROB}(L,J)) \]
Approximate Model for Arbitrary $P_{ij}$, $t_{p-1w}$, $m_{2n}$

```
DO 60 I=1,NPC
DO 60 J=1,NMP
QPR(I,J)=0.0
DO 65 L=1,NPC
IF(L.EQ.I)GO TO 65
QPR(I,J)=PROB(L,J)*((1-Y(L,J))+QPR(I,J))
65 CONTINUE
QPR(I,J)=(1.0-Y(I,J))*QPR(I,J)+1.0
QPR(I,J)=QPR(I,J)/Y(I,J)
DO 85 J=1,NMP
TEMP=QPR(I,J)
85 CONTINUE
QPR(I,J)=QPR(I,J)/TEMP
DO 88 I=1,NPC
88 CONTINUE

FORMAT(1X,'Q(M2,V.I2.),F9.5)

C
C COMPUTE UER BASED ON QUEUEING FREQUENCIES
C
DO 355 J=1,NMP
RATE(J)=1.0
DO 35 I=1,NPC
35 RATE(J)=(1.0-QPR(I,J))XRATE(J)
355 RATE(J)=1.0-RATE(J)
DO 36 J=1,NMP
36 UER(IJK)=UER(IJK)+RATE(J)
C
9999 CONTINUE
XMAX=1.0+MINP
C
PLOT RESULTS
C
CALL PLOT(21,XMAX,UER)
WRITE(19,332),NPC,NMP
332 FORMAT(///,1X,'NUMBER OF PROCESSORS = ',2X,I2,/,1X,'NUMBER OF MEMORIES = ',2X,I2,/) DO 200 I=1,21
200 WRITE(19,331),ALPHA(I),UER(I)
331 FORMAT(1X,'ALPHA=',F5.3,5X,'UER=',F8.5)
STOP
END
C
PLOTTING SUBROUTINE
C
SUBROUTINE PLOT(LIMIT,XMAX,'VALUE')
```
Appendix A-4
Approximate Model for Arbitrary Pij, tp=tw, m>n

```
DIMENSION VALUE(100)
DIMENSION J(101)
J(100)="'
C
WRITE(19,2)
2 FORMAT(11X,'
1

DO 10 I=1,LIMIT
IX=100.0*VALUE(I)/XMAX+0.5
ILAST=IX
DO 5 K=1,IX
5 J(K)="'
J(IX+1)="x'
J(1)="'
WRITE(19,11),(J(L),L=1,IX+1)
11 FORMAT(11X,181(A1))
WRITE(19,59)
59 FORMAT(/)
10 CONTINUE
RETURN
END
```
APPENDIX A-5

**********************************************************************

APPROXIMATE MARKOV CHAIN MODEL FOR TP=TW+TC

**********************************************************************

DIMENSION S(17,17),CM(16,16),TRANS(17,17)
DIMENSION Z(17),B(17)
COMMON TRANS,B,Z,NSTATE

C

TYPE 1973
1973      FORMAT(1X,'NUMBER OF PROCESSORS',/) 
ACCEPT 1974,N
1974      FORMAT(I)

TYPE 1975
1975      FORMAT(1X,'NUMBER OF MEMORIES',/) 
ACCEPT 1974,M
XM=1.0xM
K=MINT(N,M)
NSTATE=K+1

C

IF(K.NE.N)GO TO 100
TRANS(NSTATE,1)=1.0
11=1
CONTINUE

100
DO 300 I=1,17
    S(I,1)=0
300
   S(I,1)=1

C

DO 400 I=1,16
   DO 400 J=1,1
400    S(I+1,J+1)=1.0xJxS(I,J+1)+S(I,J)

C

DO 500 I=1,16
   CM(I,1)=1
500    CM(I,J)=CM(I,J-1)x1.0x(I-J+1)

C

DO 1000 I1=11,NSTATE
C
II....INDEX TO THE STATE DURING CURRENT CYCLE
Approximate Markov Chain Model for $t_p=t_w+t_c$

C I=I+I+N-K-1
C I.....NUMBER OF PROCESSORS MAKING NEW REQUEST IN CURRENT CYCLE
C
C X=X*M
C KK=MINB(I,M)
C DO 900 JJ=I,KK
C
C JJ.....NUMBER OF BUSY MPS IN CURRENT CYCLE
C J.....NUMBER OF PCS MAKING NEW REQUEST DURING NEXT CYCLE
C J1.....INDEX TO STATE AT NEXT CYCLE
C
C J=N-JJ
C J1=J-N+STATE
C TRANS(J1,I,,I)=1.0*CM(I,JJ)ES(I+J,JJ+I)/X
C
900 CONTINUE
C
1000 CONTINUE
C
DO 8000 I=1,STATE
C TRANS(I,,I)=TRANS(I,,I)-1.0
C
8000 TRANS(STATE,,I)=1.0
C
B(STATE)=1
C
CALL GAUSS
C
DO 2000 I=I+1,STATE
C I=I+I+N-K-1
C UER=UER+XM*(1.0-(1.0-1.0/XM)*I)*Z(I)
C
2000 CONTINUE
C
TYPE 1, UER
1 FORMAT(1X,F10.6)
C
STOP
END
C
SUBROUTINE GAUSS
C THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
C
C A*X=B
C
C DIMENSION A(17,17), B(17), X(17)
C COMMON A,B,X,N
Appendix A-5
Approximate Markov Chain Model for $t_p-tw+tc$

integer $S$

$S=N$

continue

if ($S-1=50,50,10$

continue

if ($S.GT.2$) go to 105

d=-$A(1,1)*A(2,2)-A(1,2)*A(2,1)$

if ($ABS(D).GT.8.0005$) go to 105

type 25

go to 100

105

do 20 $I=1,S$

$m=S-I+1$

if ($ABS(F(M,S)).GT.8.0005$) go to 30

Continue

Type 25

format (1x, 'the coefficient: t matrix is singular', ')'

go to 100

30

continue

if ($M.EQ.S$) go to 40

t=$B(S)$

$b(S)=B(M)$

$b(M)=T$

do 35 $J=1,S$

t=$A(S,J)$

$a(S,J)=A(M,J)$

$a(M,J)=T$

35

continue

40

continue

do 45 $I=1,S-1$

$k=S-1$

if ($ABS(A(S,S)).GT.8.0005$) go to 42

go to 100

42

$b(k)=b(k)-b(s)*a(k,s)/a(s,s)$

do 45 $J=1,S-1$

$a(k,J)=a(k,J)-a(k,s)*a(s,j)/a(s,s)$

45

continue

$s=S-1$

go to 5

50

continue

do 78 $I=1,N$

sum=$B(1)$

do 60 $J=1,I-1$

sum=sum-a(i,j)*x(j)

60

continue

if ($ABS(A(1,1)).GT.8.0005$) go to 61
Appendix A-5
Approximate Markov Chain Model for tp-tw tc

GO TO 100
61 x(i)=sum/a(i,1)
70 continue
100 continue
return
end
APPROXIMATE MARKOV CHAIN MODEL FOR TP > TW

\[
\text{PROB}(TP = TW + 1 + TC) = \beta \alpha \text{ALPHA/1}
\]

\[
E(TP) = TW + TC \cdot \alpha / \beta
\]

\[
\alpha + \beta = 1
\]

DIMENSION S(17,17), CM(16,16), TRANS(17,17)
INTEGER COMB(16,16)
DIMENSION Z(17), B(17)
INTEGER D
COMMON TRANS, B, Z, NSTATE

TYPE 1973

1973 FORMAT (1X, 'NUMBER OF PROCESSORS',/) ACCEPT 1974, N
1974 FORMAT (I)

TYPE 1975

1975 FORMAT (1X, 'NUMBER OF MEMORIES',/) ACCEPT 1974, M
1976 FORMAT (I, 'M = 1.0', M

TYPE 1976

1976 FORMAT (1X, 'ALPHA : PROB OF ONE MORE CYCLE OF EXECUTION',/) ACCEPT 1977, ALPHA
1977 FORMAT (F)

\[
\beta = 1.0 - \alpha
\]

NSTATE = N + 1

DO 300 I = 1, 17
S(I, 1) = 0
300 S(I, 1) = 1

DO 400 I = 1, 16
DO 400 J = 1, I
400 S(I+1, J+1) = 1.0 \times J \times S(I, J+1) + S(I, J)

DO 500 I = 1, 16
CM(I, 1) = 1
Appendix A-6
Approximate Markov Chain Model for Geometric tp

DO 500 J=2,1
500 CM(I, J)=CM(I, J-1)*1.0*(I-J+1)

C
COMB(1, 1)=1
DO 20 K=2,16
COMB(K, 1)=K
L=MIN0(K, I6)
DO 10 I=2, L
10 COMB(K, I)=COMB(K, I-1)*((K-I+1)/I)
20 CONTINUE
C

DO 1000 I=1, N
X=X*M*I
C
K IS THE MAX. NO. OF OCCUPIED MP'S
C D IS THE NUMBER OF OCC. MP'S
C
K=MIN0(I, M)
DO 1000 D=1, K
C
I-D PC ARE LEFT IN MP QUEUES
C NN PC ARE TO BE REASSIGNED TO PC OR MP QUEUES
C XPROB IS THE PROB OF D MP'S BEING BUSY
C
XPROB=CM(M, D)*S(I+1, D+1)/X
900 NN=N-I+D
TRANS(I-D+1, I+1)=TRANS(I-D+1, I+1)+XPROB*ALPHAvNN
IF(NN.EQ.0)GO TO 1000
C
DO 1000 NEUPC=1, NN
C
NEWPC= NO. OF NEW PC'S TO BE ASSIGNED TO MP QUEUES
C NEWMP= TOTAL NO. OF PC'S THAT MAKE MP REQ. NEXT CYCLE
C
NEWMP=I-D+NEWPC
C
TEMP=1.0*COMB(NN, NEWPC)*(BETAvNEWPC)*(ALPHAv(NN-NEWPC))
TRANS(NEWMP+1, I+1)=TRANS(NEWMP+1, I+1)+XPROB*TEMP
1000 CONTINUE
C
DO 1500 I=1, NSTATE
1500 TRANS(I, 1)=TRANS(I, 2)
DO 8000 I=1, NSTATE
TRANS(I, 1)=TRANS(I, 1)-1.0
8000 TRANS(NSTATE, 1)=1.0
Appendix A-6
Approximate Markov Chain Model for Geometric \textit{tp}

C
B(NSTATE)=1
CALL GAUSS
C
DO 2000 I=1,N
UER=UER+X**1.8-(1.0-1.0/XM)**1.8
Z(I+1)=CONTINUE
C
TYPE 1,UER
1 FORMAT(1X,F10.6)
C
STOP
END
C
SUBROUTINE GAUSS
C
THIS SUBROUTINE SOLVES SIMULTANEOUS LINEAR EQUATIONS
A\times X=B
C
DIMENSION A(17,17),B(17),X(17)
COMMON A,B,X,N
INTEGER S
C
S=N
CONTINUE
IF(S-1) 50,50,10
10 CONTINUE
IF(S.GT.2)GO TO 105
D=A(1,1)**A(2,2)-A(1,2)**A(2,1)
IF(ABS(D).GT.0.0005)GO TO 105
TYPE 25
GO TO 100
105 DO 10 I=1,S
M=S-I+1
IF(ABS(A(M,S)).GT.0.0005)GO TO 30
20 CONTINUE
TYPE 25
25 FORMAT(1X,'THE COEFFICIENT MATRIX IS SINGULAR','/)
GO TO 100
30 CONTINUE
IF(M.EQ.S)GO TO 40
T=B(S)
B(S)=B(M)
B(M)=T
DO 35 J=1,S
Approximate Markov Chain Model for Geometric lip

\[ T = A(S, J) \]
\[ A(S, J) = A(M, J) \]
\[ A(M, J) = T \]

35 CONTINUE
40 CONTINUE
DO 45 I = 1, S - 1
K = S - I
IF (ABS(A(S, S)) .GT. 0.0005) GO TO 42
GO TO 100
42 B(K) = B(K) - B(S) * A(K, S) / A(S, S)
DO 45 J = 1, S - 1
A(K, J) = A(K, J) - A(K, S) * A(S, J) / A(S, S)
45 CONTINUE
S = S - 1
GO TO 5
50 CONTINUE
DO 70 I = 1, N
SUM = B(I)
DO 60 J = 1, I - 1
SUM = SUM - A(I, J) * X(J)
60 CONTINUE
IF (ABS(A(I, I)) .GT. 0.0005) GO TO 61
GO TO 100
61 X(I) = SUM / A(I, I)
70 CONTINUE
100 RETURN
CONTINUE
END
CACHE MODEL FOR TP> TW, CONSTANT.

APPENDIX A-7

TYPE 1
ACCEPT 4, TP
1 FORMAT (1X, 'ENTER VALUE OF TP, FORMAT F', /)

TYPE 2
ACCEPT 4, TF
2 FORMAT (1X, 'ENTER VALUE OF TF', /)

TYPE 5
ACCEPT 4, TC
3 FORMAT (1X, 'ENTER VALUE OF TC', /)

TYPE 6
ACCEPT 4, TW
4 FORMAT (F)

TYPE 1973
FORMAT (1X, 'NUMBER OF PROCESSORS, INTEGER FORMAT', /)
ACCEPT 1974, N

TYPE 1974
FORMAT (I)

TYPE 1975
FORMAT (1X, 'NUMBER OF MEMORIES', /)
ACCEPT 1974, M
XN=1.0*M
XM=1.0*M
TA=TC-TW
DO 2000 1=0, 9
ALPHA=0.1*I
BETA=1.0-ALPHA
CONS=ALPHA*(TP+TF)+BETA*(TP-TW)

PMAX=1.0
PMIN=0.0
P=0.5

10 PNEW=1.0-CONS*(1.0-(1.0-P/XM)*XN)*XM/XN/TC/BETA
DEL=P-PNEW
DEL=ABS(DEL)
IF (DEL.LE.0.0001) GO TO 1000
IF (PNEW.GT.P) PMIN=P
IF (PNEW.LT.P) PMAX=P
P=0.5*(PMIN+PMAX)
GO TO 10
Appendix A-7

Cache Model for tpiuw

\[ C \]

1000 \[ P = 0.5^{\alpha} (PNEW + P) \]

\[ \text{UER} = 1.0 - (1.0 - P/XM) \times N \]

\[ \text{UER} = \text{UER} \times \beta _. \]

2000 \[ \text{TYPE 20, UER} \]

20 \[ \text{FORMAT(1X, 'EXECUTION RATE=', F8.4)} \]

STOP END
APPENDIX B

MULTIPROCESSOR SYSTEM SIMULATOR

This simulator can be used for multiprocessor systems with NPC processors and NMP memory units connected by a single crosspoint switch. Pc[i] has a probability PROB(i,j) of requesting service from memory j. The matrix PROB can be carefully chosen to model partially assigned crosspoints and private caches e.g. if memory unit 1 is a private cache dedicated to Pc[1], then PROB(1,1)=Prob(Pc[1] hits cache) and PROB(K,1)=0 for K≠1.

The simulation program consists of four main parts:

(i)  Main Program
(ii) Subroutine PC(ID)
(iii) Subroutine MP(ID)
(iv) BLOCK DATA

The main program contains the scheduler, which calls the two subroutines when a processor or memory is activated. This is a discrete event simulator where the two subroutines are analogous to activities in SIMULA. The scheduling is done by maintaining a doubly linked list called the sequencing set or SQS. Figure B-1 shows the structure of the SQS array.
<table>
<thead>
<tr>
<th>Successor</th>
<th>Predecessor</th>
<th>Scheduled Activation Time</th>
<th>Event Type</th>
<th>Event Identification Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure B-1  Structure of the SQS array
The first element or the head of the SQS is a dummy element, whose successor is the first real event notice. Thus, when the SQS is empty (i.e. no event notices) the first element of the SQS points to itself. The SQS is maintained in proper time order with the next most imminent event as the successor of the head.

Figure B-2 depicts a simple flow chart for the main program. The program executes in a loop until a preset simulation time is reached. At this stage it jumps out of the loop and operates on the statistics collected and outputs parameters like the number of instructions executed, the execution rate, and the queue lengths and waiting times for the various memories. A typical output is shown in fig. B-3.

The processor activity is characterized by a subroutine PC(ID); the flow chart is depicted in fig B-4. A uniform random number with range (0,1) is compared with the access probabilities in the array PROB to select a memory. A request is entered into the queue corresponding to that memory unit. An event notice for activating the memory is entered into the SQS if necessary.

Figure B-5 illustrates the working of the subroutine MP(ID). In its current version, a processor is selected as per FIFO discipline. The processor is then scheduled to be activated after a time interval equal to the sum of the memory access time(ta) and the processor's execution time(tp). The processing time distribution can be arbitrarily chosen. The program listed here has an
Initialize List Pointers and other variables

Enter event notices for all the processors at TIME=0 into the SQS.

Examine the SQS and remove the first event scheduled. Update TIME.

Is TIME greater than preset maximum?

Y

Activate the current event by transferring control to the subroutine characterizing that event.

N

Output Statistics

Figure B-2 Flow chart of the main program.
NO. OF UNIT INSTR EXEC IN 500000 TIMEUNITS IS
UNIT EXEC RATE= 0.006930

UNIT INSTRUCTIONS EXECUTED BY INDIVIDUAL PROCESSORS
868
869
864
864

ACCESS FREQUENCIES

0.297235 0.381336 0.321429
0.303797 0.331415 0.364787
0.383241 0.365741 0.331819
0.312500 0.343750 0.343750

****** MEMORY UNIT 1 ******
NO. OF REFS= 1854
MAX Q LENGTH= 2
AVG. Q LENGTH= 0.01467
AVG. WAITING TIME= 6.9573

****** MEMORY UNIT 2 ******
NO. OF REFS= 1232
MAX Q LENGTH= 2
AVG. Q LENGTH= 0.02154
AVG. WAITING TIME= 8.7419

****** MEMORY UNIT 3 ******
NO. OF REFS= 1179
MAX Q LENGTH= 2
AVG. Q LENGTH= 0.02107
AVG. WAITING TIME= 8.9372

AVERAGE VALUE= 0.02188
AVERAGE VALUE= 0.02464
AVERAGE VALUE= 0.02358

Figure B-3 Typical Simulator Output
exponentially distributed processing time. The memory reschedules itself at the end of the cycle if any request are pending in its queue.

The BLOCK DATA contains input parameters for the simulation.

This simulator could have been written in SIMULA. However, the scheduling involved does not necessitate all the capabilities of SIMULA. Moreover, the easy access to FORTRAN on the PDP-10’s time-sharing system was an important consideration in its selection.
Select the memory to be accessed.

Enter request into the queue of the selected memory.

Does the memory have any pending request?

Y \rightarrow RETURN

N \rightarrow Has the memory recovered from the previous access?

Y \rightarrow Enter an event notice for the memory at the end of its current cycle.

RETURN

N \rightarrow Enter an event notice for the memory at the current value of TIME.

RETURN

Figure B-4 Flow chart of the processor's activity.
Identify the first processor request in the queue.

Enter an event notice for the processor at 
TIME= memory access time + processing time.

Is there any other request in the queue?

Y

Enter an event notice for itself at the end of this memory cycle.

RETURN

N

RETURN

Figure B-5 Flow chart of the memory’s activity.
PROCESSING TIME --- EXPONENTIAL

C THIS PROGRAM SIMULATES A MULTIPROCESSOR WITH A CROSS-POINT BETWEEN PROCESSORS & MEMORIES

INTEGER SQS(180,5), EMPTY(99)
INTEGER SUC,PRED,SCHED,EVTYPE,EVID,EVNUM,TOPMT
COMMON/SQS/ SQS,EMPTY, SUC,PRED,SCHED,EVTYPE,EVID,TOPMT
INTEGER TA(50), TC(50), TP (50)
INTEGER TIME, SIMTIM
COMMON/GLOBAL/ TA, TC, TP, TIME, NPC, NMP, SIMTIM, NINSTR
DIMENSION COUNT(50,50), INSTR(50)
COMMON/MEM/COUNT, INSTR
DIMENSION PROB(50,50)
COMMON/PROC/ PROB, ISEED
INTEGER TQ(50), TLAST(50), LENGTH(50), MAXQL (50)
COMMON/STAT/TQ, TLAST, LENGTH, MAXQL
INTEGER QMP (50,50), FIRST (50), NEXT (50), NEXTAC (50)
COMMON/Q/ QMP, FIRST, NEXT, NEXTAC
COMMON/EXP/15D

C INITIALIZE SQS AND EMPTY LIST
DIMENSION RATE(50), JMP (50), JPC(50)
DIMENSION OPR(50,50), TEMP(50)
DATA SUC, PRED, SCHED, EVTYPE, EVID/1, 2, 3, 4, 5/
DATA TOPMT/1/
DO 21 I=1, 99
21 EMPTY(I)=1
DO 22 M=NMP
22 FIRST(I)=1
SOS(I,SUC)=1
SOS(I,PRED)=1

C ACTIVATE ALL PROCESSORS AT TIME=0
DO 23 I=1, NPC
23 CALL INSERT(I,1,0)

C CONVERT ACCESS PROBS. TO CUMULATIVE ACCESS PROBS.
DO 24 I=1, NPC

24 PROB(I, J)=PROB(I, J)+PROB(I, J-1)

C THIS IS THE SCHEDULER

C REMOVE 'FIRST' ELEMENT IN SQS AND RESTORE PERS.
1000 I=SOS(1,SUC)

1001 FORMAT(1X,'EVENT LIST EMPTY')
J=SOS(1,SUC)
SOS(1,SUC)=J
SOS(1,PRED)=1

C UPDATE TIME
TIME=SOS(I, SCHED)
IF(TIME GE SIMTIM) GO TO 2001
EVNUM=SOS(I, EVTYPE)
ID=SOS(I, EVID)

C UPDATE EMPTY LIST
TOPMT=TOPMT-1
EMPTY(TOPMT)=1

C ACTIVATE CURRENT EVENT
GO TO (1,2) EVNUM
TYPE 702
FORMAT(1X,'SCHEDULER HAS UNKNOWN EVENT TYPE')
STOP
1 CALL PC(ID)
GO TO 1800
2 CALL MP(ID)
GO TO 1800
C OUTPUT STATISTICS
2001 CONTINUE
DO 2002 I=1,NMP
2002 NINSTR=NINSTR+INSTR(I)
TIME=SIMTIM
WRITE(19,41),TIME,NINSTR
41 FORMAT(1X,'NO. OF UNIT INSTR EXEC IN',2X,112,2X,'TIME
UNIT IS',2X,112)
UER=NINSTR/TIME*1.0
WRITE(19,40),UER
TYPE 40,UER
40 FORMAT(1X,'UNIT EXEC RATE=',F10.6)
C FIND NO. OF INSTR EXEC BY EACH PROCESSOR
DO 31 I=1,NPC
DO 31 J=1,NMP
31 JPC(I)=JPC(I)+COUNT(I,J)
C FIND NO. OF ACCESSSES TO EACH MEMORY
DO 32 I=1,NMP
DO 32 J=1,NPC
32 JMP(I)=JMP(I)+COUNT(J,I)
WRITE(19,42),(JPC(I),I,1,NPC)
42 FORMAT(1X,'UNIT INSTRUCTIONS EXECUTED BY INDIVIDUAL
1 PROCESSORS',/,$0(1X,118/,))
C FIND ACCESS FREQUENCIES FOR THIS SIMULATION RUN
DO 33 I=1,NPC
DO 33 J=1,NMP
33 PROB(I,J)=COUNT(I,J)/JPC(I)
WRITE(19,443)
443 FORMAT(1X,'ACCESS FREQUENCIES',/)
DO 333 I=1,NPC
333 WRITE(19,443),(PROB(I,J),J=1,NMP)
43 FORMAT(1X,8(F8.6,2X))
C COMPUTE AVG WAIT TIME, AVG Q LENGTH
DO 34 I=1,NMP
AVQL=1.0*TQ(I)/TIME
AVWT=1.0*TQ(I)/JMP(I)
34 WRITE(19,444),I,JMP(I),MAXQL(I),AVQL,AVWT
44 FORMAT(1X,'****** MEMORY UNIT ',12,' ******',/,$0(1X,112/,1X,'NO. OF REFS=',112/,1X,'MAX Q LENGTH=',12/,1X,'AVG. Q LENGTH=',F8.5/,1X,'AVG. WAITING TIME=',F10.4)
N=TIME/10
DO 881 I=1,NMP
AV=INSTR(ID)/TIME*10.0
881 WRITE(19,282),AV
282 FORMAT(1X,'AVERAGE VALUE=',F10.5)
STOP
END
SUBROUTINE PC(ID)
DIMENSION PROB(50,50)
COMMON/PROC/ PROB,ISEED
INTEGER QMP(50,50),FIRST(50),NEXT(50),NEXTAC(50)
COMMON/O/ QMP, FIRST, NEXT, NEXTAC
INTEGER TA(50),TC(50),TP(50)
INTEGER TIME,SIMTIM
COMMON/GLOBAL/ TA, TC, TP, TIME, NPC, NMP, SIMTIM, NINSTR
INTEGER TTQ(50), TLAST(50), LENGTH(50), MAXQL(50)
COMMON/STAT/TTQ, TLAST, LENGTH, MAXQL

C PROB(I,J) = PROBABILITY(PC(I) ACCESSES PC(J))
C GENERATE A UNIFORI1 RV & DETERMINE MEMORY TO BE ACCESSED
1 RV=UNIRAN(ISEED)
RV=RV+1.0/NPC*ID
IF(RV.GT.1.0)RV=RV-1
DO 10 I=1,NMP
IF(RV.LE.PROB(ID,I))GO TO 100
10 CONTINUE
GO TO 1

100 IMP=1
C INCREMENT Q MEASURE COUNTERS
TTQ(IMP)=TTQ(IMP)+(TIME-TLAST(IMP))\times LENGTH(IMP)
LENGTH(IMP)=LENGTH(IMP)+1
IF(LENGTH(IMP).GT.MAXQL(IMP))MAXQL(IMP)=LENGTH(IMP)
TLAST(IMP)=TIME
C PUT IN REQUEST IN QUEUE FOR CHOSEN MEMORY
QMP(IMP,NEXT(IMP))=ID
NEXT(IMP)=NEXT(IMP)+1
IF(NEXT(IMP).GT.NPC)NEXT(IMP)=1
C CHECK IF MEMORY UNIT HAS OTHER REQUESTS QUEUED
IF(LENGTH(IMP).GT.1)RETURN
C CHECK IF MEMORY IS READY TO GRANT THIS ACCESS NOW
IF(TIME.GT.NEXTAC(IMP))GO TO 200
CALL INSERT(2,IMP,NEXTAC(IMP))
RETURN
200 CALL INSERT(2,IMP,TIME)
RETURN
END

SUBROUTINE MP(ID)
INTEGER QMP(50,50),FIRST(50),NEXT(50),NEXTAC(50)
COMMON/O/ QMP,FIRST,NEXT,NEXTAC
INTEGER TA(50),TC(50),TP(50)
INTEGER TIME,SIMTIM
COMMON/GLOBAL/ TA, TC, TP, TIME, NPC, NMP, SIMTIM, NINSTR
DIMENSION COUNT(50,50), INSTR(50)
COMMON/MEM/COUNT, INSTR
INTEGER TTQ(50), TLAST(50), LENGTH(50), MAXQL(50)
COMMON/STAT/TTQ, TLAST, LENGTH, MAXQL
COMMON/EXP/ISEED
IF(LENGTH(ID).EQ.0)TYPE 703
703 FORMAT(IX,'MEMORY ACTIVATED WITHOUT SCHEDULED REQUEST') .
C INCREMENT Q MEASURE COUNTERS
INSTR(ID) = INSTR(ID) + 1
TTQ(ID) = TTQ(ID) + (TIME - TLAST(ID)) x LENGTH(ID)
LENGTH(ID) = LENGTH(ID) - 1
TLAST(ID) = TIME

C IDENTIFY PC TO BE SERVICED AS PER FIFO STRATEGY
IPC=QMP(ID,FIRST(ID))
FIRST(ID) = FIRST(ID) + 1
COUNT(IPC,ID) = COUNT(IPC,ID) + 1
IF (FIRST(ID),GT,NPC) FIRST(ID) = 1

9374 RV = UNIRAN(ISEED)
TPROC = 450 x TP(IPC) x ALOG(RV)
IF (TPROC,GT,18 x TP(IPC)) GO TO 9374
JTIME = TIME + TA(IO) + TPROC

C SCHEDULE PC
CALL INSERT(1,IPC,JTIME)
NEXTAC(ID) = TIME + TC(ID)
IF (LENGTH(ID),EQ,0) RETURN
CALL INSERT(2,ID,NEXTAC(ID))
RETURN
END

C

SUBROUTINE INSERT(JTYPE,ID,TSCHED)
C THIS ROUTINE INSERTS ELEMENT I AFTER K WRT TIME
INTEGER TSCHED
INTEGER SQS(188,5),EMPTY(99)
INTEGER SUC,PRED,SCHED,EVTYPE,EVID,EVNUM,TOPMT
COMMON/SQSET/, SQS,EMPTY,SUC,PRED,SCHED,EVTYPE,EVID,TOPMT
380 IF (TOPMT,EQ,0) TYPE 794
784 FORMAT(1X,'SCHEDULER OVERFLOW')
IF (SQS(1,SUC),EQ,1) GO TO 320
J = SQS(1,SUC)
310 CONTINUE
IF (SQS(J,SCHED),GT,TSCHED) GO TO 330
IF (SQS(J,SUC),EQ,1) GO TO 340
J = SQS(J,SUC)
GO TO 310
320 K = 1
GO TO 100
330 K = SQS(J,PRED)
GO TO 100
340 K = J
100 I = EMPTY(TOPMT)
SQS(I,SUC) = SQS(K,SUC)
J = SQS(K,SUC)
SQS(I,PRED) = K
SQS(K,SUC) = I
SQS(J,PRED) = I
SQS(I,SCHED) = TSCHED
SQS(I,EVTYPE) = JTYPE
SQS(I,EVID) = IO
TOPMT = TOPMT + 1
RETURN
END
REAL FUNCTION UNIRAN(JSEED)
JSEED=JSEED*3141592621
JSEED=JSEED+7261067113
IF (JSEED)1,2,2
1 JSEED=JSEED+34359738367+1
2 TEMP=JSEED
UNIRAN=TEMP*0.2910380346E-10
RETURN
END