Performance Analysis of a Multistage Packet-Switched Shared-Memory Multiprocessor System with Restricted Outstanding Memory Requests

by

Chin Bin Wang

A dissertation submitted to the Graduate Faculty in Computer Science in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York

1996

UMI Number: 9618113

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

This manuscript has been read and accepted for the Graduate Faculty in Computer Science in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

Oct 26, 1995

Date

Chairman of Examining Committee:
Dr. Theodore Brown, Professor, Department of Computer Science, The Graduate School of City University of New York.

Nov 15, 1995

Date

Executive Officer
Dr. Stanley Habib, Professor, Department of Computer Science, The Graduate School of City University of New York.

Professor Seyed-Ali Ghozati

Professor Stanley Habib

Professor Victor Lu

Supervisory Committee

The City University of New York

Abstract

Performance Analysis of a Multistage Packet-Switched Shared-Memory Multiprocessor System with Restricted Outstanding Memory Requests

by

Chin Bin Wang

Advisor : Professor Theodore Brown

An approximation algorithm is designed for the performance analysis of a finite-buffered, packet-switched, asynchronous multistage multiprocessor system with restricted outstanding memory request. We first model the proposed machine by an open queuing network with restricted capacity. We then convert the open queuing network with restricted capacity into a corresponding closed queuing network and analyze the performance of this equivalent closed queuing network. Our algorithm is based on a decomposition method. We decompose the closed network into two sub-networks: sub1-network contains only a control node and sub2-network consists of the rest of nodes in the closed network. The sub2-network contains a finite number of stages through each of which messages are delivered successively in their transmissions. Blocking phenomena can occur on upstream paths preceding a full destination queue and the blocking effects of inter-stages are taken into account iteratively. The approximation algorithm for analyzing the performance of the sub2-network has two major steps: step 1 computes the performance of the network by *backward procedure*; step 2 computes the performance of the network by the *forward procedure*. Step 1 and step 2 are computed alternatively and repetitively until the difference of the performance measurement between step 1 and step 2 reach a small prescribed tolerance value. The throughput obtained from the sub2-network

is considered to be the arrival rate entering the sub1-network. Then we aggregate the two sub-networks to derive the average queue length, average waiting time and average server utilization of the proposed machine. We verify the accuracy of our approximate algorithm by comparing to the result of simulation method. Our verifications show that our approximate algorithm is fairly good. Scale relationships for the various performance measurements affected by a number of parameters:the number of processors, the characteristic of processors, the number of memory modules, the characteristics of memory, the topology and characteristics of the interconnection network are developed and studied. Furthermore, we compare the performance when *restricting* each processor to a maximum number of outstanding memory requests to the performance when *without restricting*.

Dedicated to

My parents, grandmother, brother, sisters and friends

Acknowledgments

I would like to express my sincere thanks to many people. Their comments and suggestions are extremely important to this dissertation:

I am grateful to my thesis advisor, Professor Brown, for his careful and patient supervision in the entire research period. He provided not only research ideas in the beginning of my Ph.D. study but also guidance and support when the research was in progress.

I wish to thank Dr. Victor Lu for his comments to this research. He spent a lot of time and provide many valuable ideas to this dissertation. According to his suggestions, this thesis is revised to be more precise and more readable.

I am also thankful to Dr. Stanley Habib and Dr. Ali Gozati for patiently attending my examinations.

Especially, I want to thank Miss Mei-Feng Fan. Thank you for her strongly support in many fields during my study period.

Finally, I want to express my special thanks to my parents, sisters, brother and many my friends for their financial and emotional support.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Preliminary

Shared-memory is the architecture of choice for a number of today's most prominent multiprocessor computers. The principal characteristic of a shared-memory multiprocessor system is that several processors are allowed to time share access to memory. The sharing capability is established through a network interconnecting the memory modules with the processors (see Figure 1). We call this the "interconnection network".

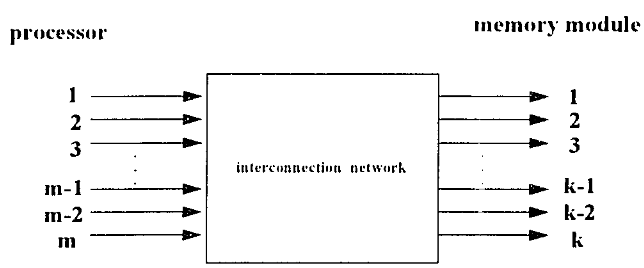


Figure 1. logical diagram

The challenge for a shared-memory multiprocessor system designer is to make the machine work faster as more processors are added - in other words, to achieve scalability. It is questionable twice as many CPUs can give twice the throughput. In fact, shared-memory machines fall far short of this idealism. Adding a second processor increases throughput often by as little as 50 percent, and this percentage drops as more processors are added. There are several factors that cause the inefficiency.

Among the factors are:

- The delay due to resource access conflicts

- Lost efficiency due to inappropriate hardware design.

- The delay introduced by the interconnection network.

- The overhead in synchronizing the work of one processor with another;

- Lost efficiency when one or more processors run out of tasks;

- The processing costs for controlling the system and scheduling operations.

Both scheduling and synchronization control are resources of overhead on a serial machine. Now we are pointing out how they degrade multiprocessor performance beyond the effects may be already on individual processors. One of the major problems for a multiprocessor system is that, as the number of processors grow beyond a certain point, the effective system bandwidth drops off due to resource access conflicts. Some studies[9,11] that have measured the performance of several existing supercomputer systems report performance degradation of over 50%. These are substantial losses for a very expensive system. In our study, we shall study how the machine's performance depends on: the characteristics of processors, the number of processors, the topology and the characteristics of the interconnection network, the characteristics of memory module and the number of memory modules.

There are two kinds of switching modes for messages transmitted between processors and memory, *circuit-switched* and *packet-switched*. In the *circuit-switched* mode, once a path is established by the address tags, the path will remain dedicated until all messages transmitted completely, after which the path is released. In the *packet-switched* mode, each message is segmented into several pieces. The unit and its destination address are combined together into a "packet". Each packet is delivered from source to destination through an interconnection network. In contrast to a circuit-switched case, each packet makes its own way from stage to stage releasing links immediately after using them. It is well known that the packet-switched mode usually yields a higher throughput than the circuit-switched mode does. This is because in case of path conflicts, blocked packets will be stored in an intermediate switch node, and will not occupy the whole path of the network.

In packet-switch mode "conflict" occur when two or more different input ports are concurrently requesting the same output port. There are two main designs, buffered and unbuffered, for handling such conflicts. With an unbuffered interconnection network, whenever a "conflict" occurs, only one packet can be accepted by the resource and others are discarded and to be sent again. Using a buffered interconnection network, conflicting packets can be stored in a buffer as long as there are buffers available. A buffered multistage interconnection network is simple to implement and also has the following advantage: The packets pass through the network stages in a pipelined way, resulting in high throughput[62]. But, real systems have buffers with finite capacities. An important phenomenon of finite buffer is *blocking*. When a finite buffer is full at one node, it could cause other upstream nodes to be blocked and this can have a significant impact on performance. It is our intention to study the performance of such a finite buffer multiprocessor machine and to see how the sensitivity of buffer size is related to the packet arrival rate and the packet transmitted rate.

Large-scale, shared-memory parallel processors use multistage, packet-switched interconnection networks for routing processor-to-memory traffic. This type of architecture should provide high bandwidth and short latency time when memory requests are distributed randomly. But, if even a small percentage of the memory requests are directed to one specific spot, the network becomes congested and its performance quickly degrades. A study by Pfister and Norton[60] shows that heavy accesses to a "hot spot" shared memory module will not only slow down those processors performing the access, but may cause the entire machine to thrash. To remedy the processors frequent access to a "hot spot" and cause the degradation of the performance of system, we propose to study if the performance of system can be improved by limiting each processor to a maximum number of outstanding memory requests.

Evaluation of performance is vital to the design and implementation of systems such as computer systems, communication systems and production systems. The success or failure of such a system is judged by the degree to which predetermined performance objectives are met. An evaluation study is always intended to answer questions about the performance of a given system. To do so, an evaluator has to gather performance information. Typically, this information consists of the values of the system's performance indices under a given workload and with specified values of the system's parameters. In general, the method of performance evaluation can be classified into three different approaches namely *simulation methodology*, *direct experiment*, and *analytical method*. There are some advantages and limitations to each method. *Simulation techniques* allow such a complex system to be studied. However, there exist some shortcomings of simulation methods, such as to determine how long to run the program in order to obtain a result near the correct value and how close the simulation estimates are to the exact value. Moreover, simulation method is a very time-consuming method. *Measurement experiments techniques* are designed for existing systems, to which they are then applied. A result of this approach is that tool designers and user frequently encounter serious problems due to the constraints imposed on the tools by the organization and implementation of the system to be measured. For a *analytical method*, one may faces an extremely complex system such that measuring the performance of system by an exact mathematical analysis becomes impossible and no reasonable approximation analysis techniques exist. To analyze the performance of such a model, several simplifying assumptions must usually be made in order to obtain a solvable model, and these assumptions may cause the model's accuracy to become unacceptable low. In our dissertation, we shall analyze the performance of our proposed model by an analytic approximation method, and we shall verify the result of our analysis by a simulation study.

Queuing network models have played an important role and are useful tool to evaluate the performance of computer systems and communication networks in the past two decades. One of the most general queuing network models which have been analyzed to date is BCMP theorem[8], combined the efforts of Baskett,Chandy,Muntz and Palacios. Their theorems show that under certain conditions a network can have a product form solution. The product form solution implies that each node in the network can be analyzed independently. Unfortunately, many features of a computer systems prevent the modeled queuing network from having a BCMP network model. Those features include the simultaneous possession of more than one resource by a customer (e.g., simultaneous possession of memory and a processor or a disk and a channel), priority scheduling disciplines, networks that have chain-dependent exponential service demand distribution at FCFS service centers or general service demand distribution at FCFS service center, and networks that have blocking phenomena due to finite capacity constrained.

Queuing network models with non-product form solutions are in general difficulty to study analytically. To analyze the performance of non-product form networks, approximation analysis techniques have been widely investigated, including analytical models and computational algorithms. For analyzing non-product form networks, the most general approximation techniques in previous studies are to decompose the queuing network into a number of sub-networks. Once the sub-networks in the queuing network are identified, each sub-network is analyzed in isolation and replaced by an equivalent node, then aggregate the results of those independent solutions to obtain an approximate solution of the original network[39,45]. Marie[50] gives an approximation method to analyze a closed queuing network with general service time distributions. The algorithm produces relatively accurate results in a very short computational time and has a consistent qualitative behavior. Dallery[16] models an open queuing networks with restricted capacity. In his work, he derives an equivalent closed queuing network model and analyzes the closed model using an approximate product-form solution technique based on Marie's method[50]. However, those studies[16,39,45,50] analyze their models by assuming that all buffers are infinite.

In the real-world, buffers are always finite. An important feature of a queue with a finite buffer is that a server may become blocked from continuing service when the capacity of its destination queue is reached. A detailed description and comparison of different blocking types has been presented in[1,56]. A number of previous studies investigate the performance of a network with finite buffers by taking into consideration the influence of blocking. Those studies include [1,2,3,7,53,54,55]. Akyildiz I.F[1,2] analyze closed queuing networks with transfer blocking. His concept is based on the fact that finite station capacities limit the number of flexible states in the network. In his algorithm, he shows to reduce the state space by considering the finite capacities of the station and derives a nonblocking queuing network with approximate total number of jobs of which the state space is equal to the state space of the blocking queuing network. In[1], Akyildiz studies a two-station, multiple server closed queuing network with blocking. In his work he shows that a two-node closed network is identical (i.e., has the same rate matrix) to a two-node closed queuing network without blocking. In[2] he develops an approximation algorithm for the throughput of closed queuing networks with exponential and general service times. He approximates the throughput of a closed queuing network with blocking by assuming that the throughput of a blocking network is approximately the same as an equivalent network with infinite capacities. The approximation algorithm takes into account only the total number of states, and the throughput estimates are insensitive to the location of nodes and the service rates. Due to its oversimplification, the relative error percentage of the algorithm[2] can easily reach as high as 25 percent[56]. In[3] he extends the mean value analysis algorithm[60] to analyze a single server queuing network with blocking. The algorithm calculates the mean queue lengths based on the modification of mean residence times due to the blocking events that occur in the network. Onvural and Perros[55] show that the equivalencies exist between open and closed queuing networks with finite buffers. They also develop an approximation algorithm to calculate the throughput of large closed exponential queuing networks with finite queues. They produce relative good accuracy at the expense of computational complexity. In[56] they show that the product form queue length distribution exists for a group of networks, when k= min(B_i, i=1,...,N)+1. This is so since there can be at most one node blocked at a time, and when a server is blocked there cannot be any customer waiting in this queue. Thus, during the blocking period, the service in the blocked node behaves like an additional space for the blocking node. However, if the populations are greater than the min $\{C_i\}+1$, we can only solve the network to obtain the approximate throughput. In[56] they also state that increases the population of a closed queuing network with finite queues, the throughput initially increases. However, after a certain population K^* the throughput begins to drop because there is greater contention for spaces by customers and blocking becomes more likely. From the point the throughput steadily declines as more customers are added. If the population that gives maximum throughput can be determined(even approximate) and the throughput when the population is maximum is known, we can fit suitable curve through those points to estimate throughput for all value of K. The crucial step in the approach is to find the population that gives the highest throughput Another approximation algorithm considering a transfer blocking queuing network is designed by Suri and Diehl[64]. The algorithm is used to compute the throughput of the network. In their work, the service time at each node is assumed to be exponentially distributed and if the networks is closed, one of the stations must have an infinite capacity. The algorithm involves the concept of *variable buffer size* and is used together with the *flow equivalent approximations*. The method gives only the throughput of the entire network, it does not give statistics for individual stations. Again, the method produces accurate result at the expense of computational complexity. In [7] S. Balsamo etal.. provides an algorithm to evaluate the cycle time distribution for cyclic closed exponential networks formed by N nodes, where each node can have finite capacity queue and is a *blocking before service-server occupied* type. Their algorithm has a low order polynomial time computational complexity in the state space of the Markov process associated with the queuing network.

A considerable number of previous studies investigate the performance of various topologies of multistage interconnection networks[21,23,30,41,44,45,57]. These studies include buffered or unbuffered, synchronous or asynchronous, and circuit-switched or packet-switched interconnection networks. However, the interconnection networks in these studies are typically modeled in isolation and, thus, the ways in which other system resources effect system performance ignored. Moreover, some of these works rely on a number of over-simplifying assumptions. For example, some rely on constant memory request generating rate or infinite buffers. Harrison and Pinto[32] give an approximation algorithm for the performance analysis of the buffered, packet-switched, asynchronous networks with no feedback. In their work, the networks are organized in a finite number of stages, through each of which a task passes successively in its transmission. Blocking can occur on several upstream paths preceding a full component and the inter-stage blocking effect is taken into account iteratively. There are some previous works[9,19,66] which analyze the entire multiprocessor system. I. Y. Bucher[9] examines both an open queuing model and a closed queuing model. In his study, he derives several scaling relationships among the processors and memory modules. However, in his model, he assumes that processors and memory modules are connected by an ideal interconnection network that does not contribute to memory access delays. D. L. Willick and D.I. Eager[66] develop an analytical model for clocked, buffered, packet-switched multistage interconnection networks that is based on queuing network models and they analyze the performance by approximating mean value analysis[**AMVA**]. In their analytical model, arbitrary network topologies and memory reference patterns are permitted, and processors may have a maximum number of outstanding memory requests. They obtain the average residence time for each customer under different types of services, compute the throughput of the overall system and the average queuing length at each servers. In their study[66], they assume that the number of packets in the network are constant and equal to the number of maximum outstanding memory requests. However, due to the situation that memory request rate is low, results of their analytical model may not be useful. Moreover, in their model[9,66], it is assumed that there is always available space in the queue for the arriving customers: i e. an infinite buffer case.

1.2 Our Proposal

In our study, we shall analyze the performance of an asynchronous, distributed control, packet-switched MIMD(multiple instruction, multiple data) shared-memory system. The proposed machine has the following characteristics amending previous studies in this area:

(1) there is a finite buffer for each memory module and for each switch element.

(2) each processor can only generate a maximum number of outstanding memory requests.

The condition (1) should make our model closer to a real machine and hence provide for more accurate measurement of performance than many previous studies.

In this dissertation, we shall employ a queuing network model to represent the proposed multiprocessor system. We shall convert the proposed queuing network model into an equivalent closed queuing network. We shall analyze the performance of the proposed multiprocessor machine based on different buffer sizes and different arrival rates. We shall study if the performance of system can be improved by restricting each processor to a maximum number of outstanding memory requests. We shall derive the approximation scaling relationships among the number of processors, the topology of interconnection network, the number of memory modules, and the memory cycle time. We shall give an approximation algorithm to derive the performance measures (throughput, average waiting time in each stage, and response time) of the proposed system. The accuracy of our approximation algorithm will be verified by a simulation method. We expect our model and its approximation algorithm can contribute toward the design of future systems.

1.3 Dissertation Outline

This dissertation consists of eight chapters. Chapter 2 describes several different interconnection networks and their characteristics. We shall discuss the fundamental decisions in determining the appropriate architecture of an interconnection network for a multiprocessor system. We also present the behavior of each major component of the proposed architecture machine, which will be studied in our dissertation.

Chapter 3 reviews some queuing theorems. In section 3.2 we introduce Jackson, Gordon-Newell and BCMP theorem and how those theorems are used to evaluate the performance of a queuing network model and discuss the limitations of these theorems.

Chapter 4 will explain how to represent our architectural model by a finite buffer open queuing network model with restricted capacity. In section 4.2 we present a technique to convert the finite buffer open queuing network with restricted capacity into an equivalent closed queuing network. In section 4.3 we discuss blocking cases and blocking types in the closed queuing network.

Chapter 5 analyzes the performance of the proposed closed queuing network model and derives the scaling relation among the number of processors, the number of memory modules, and the memory access time for our proposed model by approximation methods. We also introduce our developed the computational algorithms and demonstrate the computations of the performance measurement parameters(throughput, utilization of resource, response time and delay time).

Chapter 6 describes our simulation design and program.

Chapter 7 results of a simulation of the proposed closed queuing network model are given and compared with the results of the analytical model obtained in chapter 5. We also discuss the numerical results obtained and try to explain their significance and meanings.

Chapter 8 conclusions and future works.

Chapter 2

Architecture of Our Proposed Machine

A shared-memory multiprocessor computer consists of three major components: processors, an interconnection network and memory modules. The interconnection network plays a bridge role between processors and memory modules. With messages are exchanged through the interconnection network. Naturally, the performance of the interconnection network can have a large effect upon the overall system performance. There are several factors to affect the performance of the interconnection network. Those factors include the topology of the interconnection network, the switching mode and the methodo logy for solving resource access conflict. We, therefore, devote section 2.2 to a discussion of the topology of the interconnection network and the explanation of the fundamental decisions in determining the appropriate architecture of an interconnection network for a multiprocessor machine. In section 2.3, we shall describe the architecture of each type of the major component and its characteristics in our proposed model of a computer.

2.1 The Interconnection Network for a Multiprocessor Computer System

Various topologies of interconnection networks will be discussed in the section 2.2. The simplest way to construct a multiprocessor system is to connect the processors together using a common bus. Each processor has access to this common bus. Also, attached to this bus is the central memory which is a global resource for all processors. Once all processors access the central memory frequently, there could be severe contention on the bus. Consequently, The system could cause arbitration delays and reduce performance. To prevent this problem, the architecture is often designed for each processor to have a local memory and a cache memory. The objective of using cache and local memory is to shorten the memory cycle and reduce the use of the share memory and thereby limit the effects of contention when processors have to use shared memory. However, the bus is not the only potential bottleneck in the bus-oriented multiprocessor. The shared memory is another one. When the bus bandwidth increases, the performance of the system is eventually limited by the shared-memory due to memory access conflict. In other words, this topology's simplicity is at the expense of time complexity. Therefore, a time-shared common bus supports a very limited data transfer rate in order to avoid contention. It is, therefore, inadequate for even a small scale multiprocessor system.

At the other end of the spectrum, the interconnection network can be a crossbar network. Figure 2.1 shows a crossbar network that connects N processors with N memory modules. The communication of a crossbar network has no contention. It gives the best performance compared to any kind of interconnection networks in terms of time complexity. However, the number of gates in an NxN crossbar are proportionate to N^2. For a large system this is overly expensive because of the number of gates and the factor of reliability.

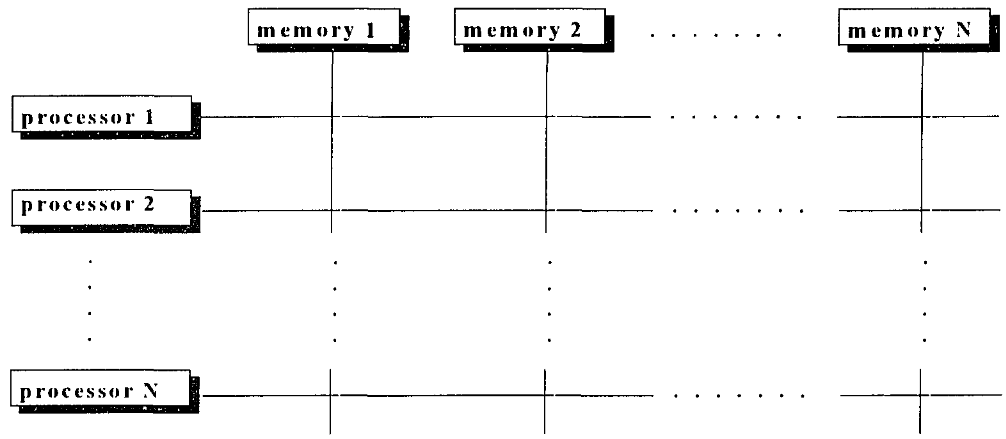


Figure. 2.1 a crossbar switch in a multiprocessor system

Between these two extremes is a multistage interconnection network(**MIN**) which provides a fast and flexible communication at a reasonable cost. The generalized multistage topology has $m = \log_2 N$ stages, where each stage consists of a set of N lines connected to N/2 interchange boxes. Each *interchange box* is a two-input, two-output device and can be set to one of four states: (1) *straight* (2) *swap* (3) *lower broadcast* (4) *upper broadcast*. A multistage interconnection network allows any input to be dynamically connected to any output if there is a free path. The cost in number of gates of N-way **MIN** grows almost linearly O(NlogN) compared with N^2 of a crossbar. However, a **MIN** could cause contention for the internal links(switch elements), that reduces the throughput of the entire system. The types of multistage interconnection networks include: a data manipulator, a flip network, an indirect binary n-cube network, an omega network, and a regular SW banyan network with spread and fan-out of 2. All are topologically equivalent[68]. In section 2.2, we shall discuss the structure of each of these.

2.2 Topology of interconnection network

The topology of a network can be defined as the actual interconnection pattern used to connect a set of N sources to a set of M destinations. The interconnection network can be divided into two categories: *static networks* and *dynamic networks*.

2.2.1 Static Networks

A *static network* has no switching elements between two processors. Links between two processors are passive and dedicated paths can't be reconfigured. Topologies in the static networks can be classified according to the dimensions required for layout.

One Dimension

Bus: in a bus topology, a large number of stations can be connected to the bus. Each of these stations can broadcast simultaneously to the others.

Two Dimensions

1). Ring: the ring topology is a sequence of point-to-point links with flow in one direction around the ring.

2). Star: a star topology requires that a high-speed link be dedicated for communications between the central 'hub" and each terminal connected to the 'hub" during periods of operation.

3). Tree: the tree topology is used to distribute data over a wider area. Typically, in networks using the tree topology, remote stations access a central processor. A ubiquitous example of such a network used multipoint private lines in common carrier networks.

4). Near-Neighbor Mesh.: In a near-neighbor mesh network, the nodes are arranged into a d-dimensional lattice. Communication is allowed only between neighboring nodes; hence interior nodes connected with 2d other nodes.

Multi-Dimensional

Hypercube: A k-dimension hypercube network consists of $N = 2^k$ nodes, forming a k dimensional network. The nodes are labeled 0, 1, 2,...2^k--1. Two nodes are adjacent if the binary representation of their labels differ in exactly one bit position.

2.2.2 Dynamic Networks

A *dynamic network* can be denoted as that which has switch boxes between resource and destination. Links in the dynamic category can be reconfigured by setting the network's active switching boxes. The dynamic network can be divided into *single stage* and *multistage* network, according to the number of switching boxes that are passed through from source to destination.

Single-Stage interconnection network

Shuffle Exchange: A *shuffle exchange* network consists of two kinds of connections, a shuffle and an exchange. The shuffle connection links node i with node [$2i$ mod (n - 1)], with the exception that node n-1 is connected to itself. The exchange connection links pairs of nodes whose numbers differ in their last significant bit. (i.e, odd & even pairs)

Crossbar Network: A *crossbar network* is an extreme case in which each input is connected to each output through a path that contains a single switching node. The topology allows parallel access from source to destination without conflict.

Multistage Interconnection Network

A multistage network consists of more than one stage of switching elements and is usually capable of connecting an arbitrary input node to an arbitrary output node. Based on the ability of the processors to transmit data concurrently, the type of multistage interconnection network can be classified into: *blocking*, *rearrange nonblocking*, and *nonblocking* multistage network.

Blocking Multistage

Baseline Network: A *baseline network* consists of logN stage. Each stage has N/2 2 x 2 crossbar switches. The connection between the first stage and the second stage is an N x N perfect shuffle connection. The second and the third stage contain two (N/2) x (N/2) perfect shuffle connection. The process can recursively be applied to the topology in each iteration until the size of $(N/2^k) \times (N/2^k)$ is equal to 2 x 2 crossbar switch.

Delta Network: An $a^n \times b^n$ *delta network* consists of n stages $S_0, S_1, \ldots S_{n-1}$. There are a^{n-1} identical a x b crossbar modules in the first stage(a^n inputs and $a^{n-1}b$ outputs). This implies that the stage two must have $a^{n-2}b$ crossbar modules and the *i*th stage has $a^{n-i}b^{i-1}$ identical a x b crossbar modules. The connection between every stage is linked by using the a-shuffle link pattern.

Indirect Binary n-cube: An *indirect binary n-cube network* consists of an m stage and each stage has N/2 two function interchange boxes($N=2^m$). The link between two stage is constructed such that the two inputs link to a box in stage *i* which differs only in its *i*th bit position.

Omega Network: An N x N *omega network* consists of $\log^N$ identical stages. Each stage is a perfect shuffle connection followed by a column of N/2 interchange boxes (see figure. 2.2).

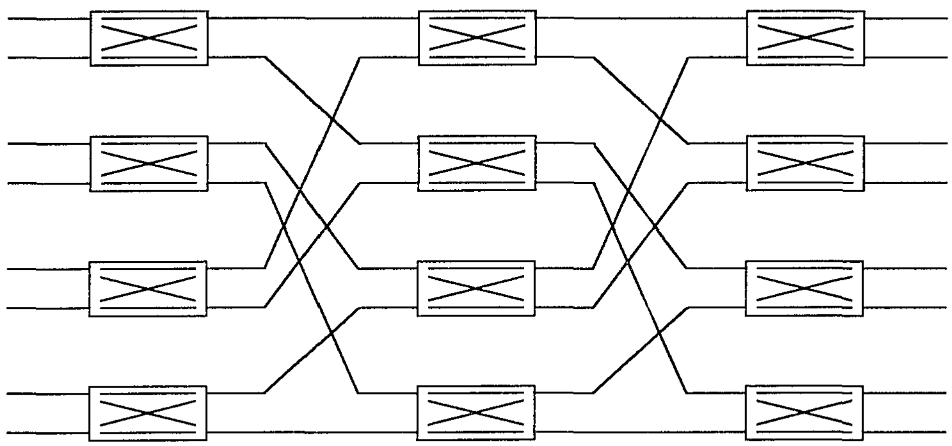


Figure 2.2 An 8 X 8 Omega interconnection network

Rearrange Nonblocking

A *rearrange nonblocking network* can be defined as any set of paths between source and destination which can be reassigned new routes(if necessary) so as to make it to perform all possible connections without blocking. An example is *Benes Network*

Benes Network: *Benes network* can be thought of as an *Omega network* that has extra hardware added to it so that there are many possible paths from each input to each output. Benes network can achieve all the permutations of nonblocking network.

Nonblocking Network

A nonblocking network is defined as: a network which can perform all possible connections without blocking. For instance, a *clos network*.

2.3 The Characteristics of an Interconnection Network

There are four fundamental decisions in determining the appropriate characteristics of an interconnection network for a multiprocessor machine. The decision factors are *timing mode*, *control strategies*, *switching methodology* and *network topologies*.

Timing mode: Timing mode can be classified into three categories: *synchronous*, *asynchronous* and *combined*. Synchronous communication is needed for establishing communication paths synchronously for either a data manipulating function or for a data instruction broadcast. For an SIMD machine, the MIN is normally used in synchronous mode. Asynchronous communication is needed for multiprocessing in which connection requests are issued dynamically. For an MIMD machine, the MIN is used asynchronously. A system may also be designed to facilitate both synchronous and asynchronous processing.

Control strategy: A typical interconnection network consists of a number of switching elements and interconnecting links. Interconnection functions are realized by properly setting the control of the switching elements. There are two approaches for switch control. The control-setting function is managed by a centralized controller, we call it *centralized control*. Another alternative control-setting function is managed by the individual switching element, this strategy is called *distributed control*.

Switching methodology: The two major switching methodologies are *circuit switching* and *packet switching*. In circuit switching, a physical path is actually established between a source and a destination before communication and is disconnected when data is completely transmitted. The classic example is the telephone system, in which a circuit is established when a call is made and remains until the call is terminated. In packet switching, data is divided into several packets and routed through the interconnection network without maintaining a physical connection path initially. In general, circuit switching is more suitable for bulk data transmission, and packet switching is more efficient for many short data messages.

2.4 The Characteristics of our Proposed Architecture

In this section, we will discuss the characteristics of each type of major component: processors, interconnection network and memory modules. These three components consist of our proposed machine. We shall analyze the performance of the proposed machine throughput our dissertation.

2.4.1 The Characteristics of Processors

We study an MIMD multiprocessor system. In an MIMD system all processors can generate memory requests independently and to require access to any memory module independently of the other requests. It is well known that frequent access to shared-memory not only slow down those processors requesting access, but may cause the entire system to thrash[59]. To avoid this, a multiprocessor system is designed for each processor to have a small local memory and a high speed cache. Whenever a processor generates a memory request, the operating system first checks the cache and local memory. If the requested data is not available in either, a requesting message is delivered through the interconnection network to the shared-memory. This topology can reduce the number of shared-memory access from processors and reduce the traffic for memory access which could lead to a degradation of the performance of the entire system. Furthermore, in our dissertation we shall study the performance of a shared-memory multiprocessor machine by restricting each processor to a maximum number of outstanding shared-memory requests. In such a system, a processor may generate a new memory request only if it's not blocked, i.e., when its outstanding requests are smaller than maximum allowed. In the case that memory request from processor i is not blocked, processor i will generate a memory request according to a *Poisson distribution*. However, when the outstanding memory requests from processor i reach the maximum number, processor i will be blocked; The blocked processor i will regenerate the memory requests after it receives a reply from memory. Let us denote NM_i to be the number of maximum outstanding requests for processor i. This step assures that the number of memory requests from each processor in the system do not exceed NM_i , i = 1,..,N.

2.4.2 The Characteristic of Interconnection Network

There are two distinct types of requests "forward" (for requests sent from processor to memory) and "return" (for replies sent from memory to processor) *Omega networks*(see figure 2.2). In the dissertation, we study the packet-switched mode. The proposed interconnection network queues packets in each output port by a finite buffer. Messages are passed stage to stage through the interconnection network. To deliver the message to the predetermined memory module, the packet requires a routing tag(destination address). The routing tag is added to the packet and is examined at each stage(i.e.,switch box). After examining the routing tag, the packet is delivered to the next stage of the destination output port. If the output port(link) of destination is not occupied, the message is transmitted to the following next stage from the output port of destination immediately. Otherwise, the packet is queued by first come first service(FCFS) discipline as long as buffer size is available. Once the number of packets in the buffer of the destination reach the maximum capacity, the previous stage is forced to stop transmitting packets until a packet is completely transmitted from the output port of destination. The mean transmit time from one stage to the following stage is assumed to be 1 time unit/per packet.

2.4.3 The Characteristic of Memory

Each memory module has an input port for receiving requests from the forward network and an output port for replying to these requests through the return network. After completing memory access, the reply message for the request is sent to the output port and, simultaneously, the next memory request in the input port will be served if there exist any request packets in the queue. Similarly to the interconnection network, the number of packets in the buffer of the destination reach its capacity, then the previous stage is forced to stop transmitting packets until a packet is serviced.

The flow of processor generate a memory request is shown as Figure2.3

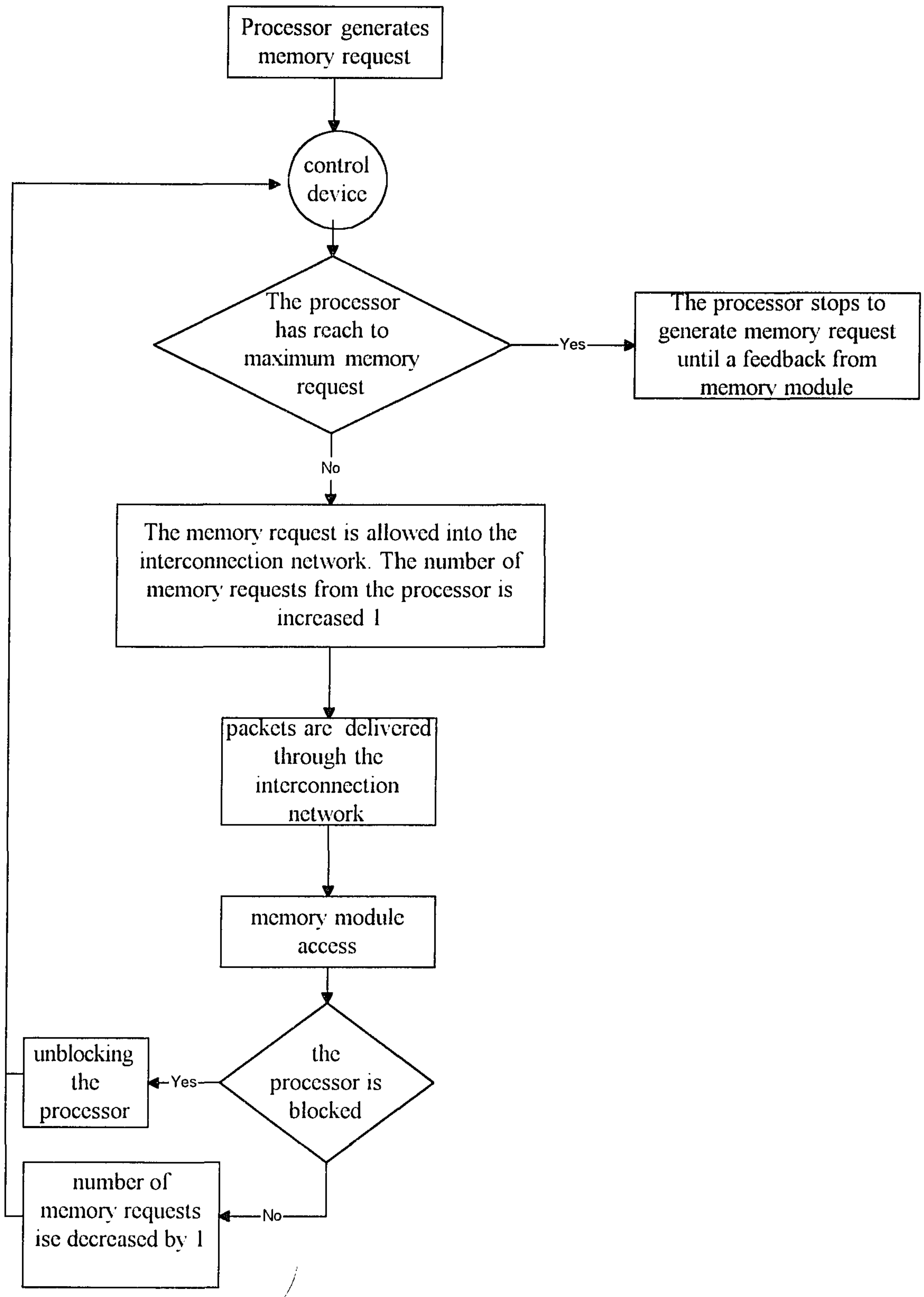


Figure 2.3 Flowchart of processsor generating memory request

Chapter 3

Queuing Network Models

In some complex systems, such as multiprogramming computer systems, switching machines, computer communication, or transportation system, customers usually require several stage of services provided by different servers and may have to wait in a sequence of queues before completing the required service.

To analyze the performance of such complex systems, we usually employ a queuing network to model the proposed system. In the years past queuing network models have played an important role and have been proved to be a powerful tool for performance analysis and prediction in these areas. In our dissertation, we shall model the proposed multiprocessor system by a queuing network model. We, therefore, devote this chapter to discuss the behaviors of queuing network models. In section 3.1 we shall describe the basic concepts of queuing system. In section 3.2 we shall describe three queuing network models open queuing network model, closed queuing network model and queuing network with restricted capacities model. In section 3.3 we shall review Jackson theorem, Gordon-Newell theorem and BCMP theorems. Their theorems show that, in the steady state distribution, some types of queuing networks can posses product form solution. This means that each node can be analyzed in isolation as if it were an independent queuing network node. Those discoveries provide a remarkable and effective way in analyzing the performance of some complex systems.

3.1 Description of The Queuing System

A queuing system can be described as customers arriving for service, waiting for service if the server is not available immediately, leaving the system after being served. To describe the behavior of a queuing system, six basic characteristics of the process need to be specified: (1) arrival pattern of customers. (2) service pattern of servers (3) service discipline (4) system capacity (5) number of servers (6) number of service stages.

Arrival Pattern of Customers

The arrival pattern is often described as *mean arrival rate*(input rate) and the *customer arrival probability distribution*. The arrival rate can be random, fixed or depends on the number of customers in the network. The external population might be finite or infinite in size. Also, a customer may decide to wait no matter how long the queue becomes, or if the queue is too long, the customer may decide not to enter it.

Service Pattern of Servers

The service pattern can also be described as mean service rate and service time probability distribution. The service rate can be random, fixed or depends on the number of customers waiting for service. An example for state-dependent service rate, a server may work faster if he sees that the queue is building up.

Queue Discipline

Queue discipline refers to the manner that customers are selected for service when a queue is formed. The most common queue may be organized as first-come,first-served(FCFS) structure. However, this is not the only queue discipline, there can be other interesting discipline to be discussed as needed.

Queue Capacity

The queue may be able to accommodate any number of customers, or have a finite capacity. With a finite capacity, a customer is forced to balk if he arrives at a time which queue size is at its maximum capacity.

Number of Servers

The number of servers can be viewed as the number of parallel service stations which can service customers simultaneously.

3.2 Type of Queuing Network Models

A *queuing network* can be thought of as a connected directed graph whose nodes represent the service centers. The arc between two nodes indicate the one-step moves that customers may make from one service center to another service center. Each node has its own queue, customers served according to some scheduling strategy. Customers may be of different types and may follow different routes through the network. Also, customers move through the network may be deterministic or random. In order to describe the behavior of a queuing network, one needs to specify the parameters of each node:service rate and time distribution, the number of servers, the queuing capacity, the queuing discipline, and the routing pattern. In general, we can classify queuing networks into three types based on the topology of the graph and the nature of the customer population: open, closed and mixed queuing network. An *open queuing network* system has at least one arc coming from outside and at least one arc going out. Customers enter the network from outside, receive services at one or more node, and leave the network eventually. For a *closed queuing network*, there is a fixed number customers circulating in the network. That is, no arrivals to or departures from the network are allowed. An *open queuing network with restricted capacity* **(OQN-RC)** is the network that specifying characterized as the number of customers, which can be simultaneously in the system at one time, can't exceed a fixed number. This fixed number is called the capacity of the network. In a OQN-RC system, a new customer can enter the system only if the number of customers currently existing in the network is smaller than the capacity of the system has.

3.3 Theorems Related to Queuing Network Model

3.3.1 Jackson Theorem

J. R. Jackson[37,38] in his two pioneering papers show that for an open queuing network consists of N nodes satisfying the following conditions:

1. Customers arrive from outside the system into node i in Poisson process.

2. Each node i consists of c_i identical server. The service time in node i is exponentially distributed with mean rate μ_i.

3. After served at node i, a customer enters node $j(j=1,2,...N)$ or leaves the network with a probabilistic choice, independently of past history.

Then the steady-state distribution of the Jackson network has the product form

$$P(n_1,n_2,\ldots,n_N) = p_1(n_1)p_2(n_2)\ldots p_i(n_N) \qquad (3.1)$$

where $P(n_1,n_2,\ldots,n_N)$ is denoted as the steady state probability that there are n_1 customers at node 1, n_2 customers at node 2,...,$n_N$customers at node N. $P_i(n_i)$ is the steady state probability that there are n_i customers in the ith node. The result implies that each node can be analyzed in isolation as if it were an independent M/M/c_i queuing system with arrival rate λ_i and service rate μ_i.

3.3.2 Gordon-Newell Theorem

Gordon-Newell Network

In modeling computer and communication networks, a closed queuing network is often more useful than an open queuing network because population is normally limited by the resource capacity. Gordon-Newell[27] extends Jackson's theorem to closed networks of exponential servers and shows a network satisfies the following conditions.

1. the network is closed and has fixed population M.

2. node i is queue length depend with service rate $\mu_i(n)$ when it has n customers.

3. After completing service at node i, a customer goes to node j with probability P_{ij}, independent of past history.

Then the closed queuing network has a product form solution.

Let $\mu_i(j)$ is the service rate of node *i* when its queue length j in a N nodes and M customers Gordon-Newell network. Then the steady state distribution of the network is given by

$$P(n_1, n_2, n_3, \ldots, n_N) = \frac{1}{G}\alpha_1(n_1)\alpha_2(n_2)\alpha_3(n_3)\ldots\alpha_k(n_N), n_i \geq 0, n_1 + n_2 + n_3 + \ldots + n_N = M \tag{3.2}$$

where

$$\alpha_i(n_i) = \frac{e_i^{n_i}}{\prod_{j=1}^{n_i} \mu_i(j)} \tag{3.3}$$

and G is the normalizing constant defined by

$$G = \sum_{n_i \in S} \prod_{i=1}^{N} \alpha_i(n_i) \tag{3.4}$$

The total customers in the network have fixed population M. It is, therefore, the state spaces of the closed queuing network model are finite and can be described as

$$S = \{(n_1, n_2, n_3, \ldots, n_N) \mid \sum_{i=1}^{N} n_i = M\} \tag{3.5}$$

3.3.3 BCMP Networks

In Jackson and Gordon-Newell networks, all customers are statistically identical. The service time and the queuing discipline are limited to be exponentially distributed and FCFS, respectively. However, in a real system, the server typically serves different types customers which can have quite different resource requirements. For example, in a computer system there are jobs which are CPU bound, and only use a very small amount I/O, as well as jobs which are I/O bound and only being processed for short durations between successive I/O requests. In order to model existing real-life complex systems we need to expand on their restrictions and allow customers of different classes to have different characteristics, such as routing behavior and service requirements. To evaluate the performance of such a complex system, BCMP theorems provide an effective solution. The class of queuing networks with a product form solution have been extended by Baskett, Chandy, Muntz and Palacios[8] to include the cases such as different classes of customers and general service distribution for certain service disciplines. The theorems allow different types of customer classes in the network and a customer may change class membership while making a transition from one station to another. Furthermore, the service discipline of a service center can be one of the following four types:*first come first served*, *processor sharing*, *infinite servers* and *last come first served preemptive resume*.

3.3.3.1 Coxian Distribution

A *coxian distribution* can be defined as a customer may go through a series of servers with exponential distribution service time with mean $\frac{1}{\mu_i}$ in each station, but with the possibility of leaving the network before entering any stage of service, as shown in Figure 3.1.

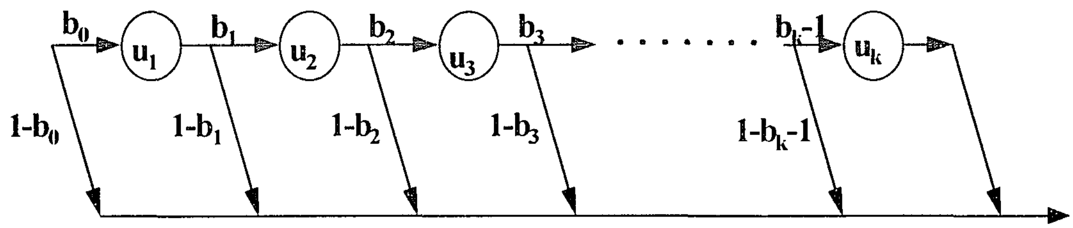


Figure 3.1 Coxian model

Let b_i to be the probability that a customer complete service at stage i entering stage i+1. Then the probability A_k, a customer enter stage k and leave network before it enter stage k+1, can be expressed as $A_k = b_0 b_1 b_2 \ldots b_{k-1}(1-b_k)$. Hence, the time π a customer spending in the network is the sum of k stages independent, exponentially distributed random variables. The expected value of π is derived as follows:

$$E(\pi)=\sum_{i=1}^{l} A_i(1-b_i)\sum_{j=1}^{i}\frac{1}{\mu_i} \qquad (3.6)$$

Coxian distributions are very general. The exponential, erlang and hyperexponential distribution are *Coxian distributions*. The sum of *Coxian distributions* is also Coxian. In fact, *Coxian distributions* have the same property as a set of distributions with rational Laplace transforms. Moreover, any probability distribution function can be constructed as closely as required by *Coxian distribution*.

3.3.3.2 Types of Service Center

In general, a service center in the network is defined by the *queueing discipline* and the *service time distribution* for each class of customer. The BCMP theorems state that a queuing network with multi-class traffic has a product form solution for the steady state when service centers are one of the following four types:

1. First come first served(**FCFS**): The service discipline requires all class customers have identical exponential service time distribution with mean $\frac{1}{\mu_i}$. Customers are served in order of arrival.

2. Processor sharing(**PS**): Each job receives a quantum of service and suspended until every other jobs have received an identical quantum of service in a round-robin fashion. In this case, we can allow customers of different class to have different service time distributions, as long as they are Coxian distributed.

3. Infinite servers(**IS**): All arrival customers receive immediate service, so that each customer is delayed only by a service time. In this case, each class of customer can have distinct service time distributions, as long as they are Coxian distributed.

4. Last come first served preemptive resume(**LCFS-PR**): As with the **IS** nodes, the arrival customer also receive immediately service. The arrival customer can go directly into service, pre-empting the customer in service(if any). After the customer completes service, the pre-

empted customer resumes service from the point of interruption. **LCFS-PR** also allow each class of customer can have distinct Coxian service time distribution.

3.3.3.3 BCMP Theorems

The BCMP theorem states that queuing network with BCMP node and multi-class traffic have a product form solution for the steady state joint probability distribution of the node states. Let the state space be denoted by $S=(S_1,S_2,\ldots,S_N)$, where S_i depends on the type of node. The general solution of the global balance equations has the form

$$P(S_1.S_2.\ldots.S_N) = \frac{1}{G} d(S) \prod_{i=1}^{N} f_i(S_i) \tag{3.7}$$

where

(a) G is the normalizing constant.

(b). if the network is open and has only 1 chain then

$$d(S) = \prod_{k=0}^{K-1} \lambda(k) \tag{3.8}$$

otherwise, if the network is open and has m chains(chain c has population K_c)

then

$$d(S) = \prod_{c=1}^{m} \prod_{k=0}^{K_c-1} \lambda_c(k) \tag{3.9}$$

if the network is closed then

$$d(S) = 1 \tag{3.10}$$

(c). The factor $f_i(S_i)$ depends on the type of node i and the number of customers present at node i.

For type 1 (**FCFS**) nodes, $S_i = (r_{i1}, r_{i2}, r_{i3}, \ldots, r_{in_i})$, where r_{ij} is the class type of the jth customer waiting in FCFS order at node i.

$$f_i(S_i) = \prod_{j=1}^{n_i} \frac{e_{ir_{ij}}}{\mu_i(j)} \tag{3.11}$$

For type 2 (**PS**) nodes, $S_i = (S_{i1}, S_{i2}, \ldots, S_{ir})$ and $S_{ir} = (n_{ir1}, n_{ir2}, \ldots, n_{irl})$, where n_{irl} is the number of customers at node i of class r and in stage l of their service.

$$f_i(S_i) = n_i! \prod_{r=1}^{R} \prod_{l=1}^{L_{ir}} \left[\frac{1}{n_{irl}!}\left(\frac{e_{ir}A_{irl}}{\mu_{irl}}\right)^{n_{irl}}\right] \tag{3.12}$$

For type 3 (**IS**) nodes, $S_i = (S_{i1}, S_{i2}, \ldots, S_{ir})$ and $S_{ir} = (n_{ir1}, n_{ir2}, \ldots, n_{irl})$, where n_{irl} is the number of customers at node i of class r and in stage l of their service.

$$f_i(S_i) = \prod_{r=1}^{R} \prod_{l=1}^{L_{ir}} \left[\frac{1}{n_{irl}!}\left(\frac{e_{ir}A_{irl}}{\mu_{irl}}\right)^{n_{irl}}\right] \tag{3.13}$$

For type 4 (**LSFS-PR**) nodes, $S_i = ((r_{i1}, l_{i1}), (r_{i2}, l_{i2}), \ldots, (r_{in_i}, l_{in_i}))$ where r_{ij} is the class of the jth customer waiting in arrival order at node i and l_{ij} is its stage of service.

$$f_i(S_i) = \prod \left| e_{ir_{ij}} \frac{A_{ir_{ij}l_{ij}}}{\mu_{ir_{ij}l_{ij}}} \right| \tag{3.14}$$

Chapter 4
Modeling the Proposed Architecture

In this chapter, we shall show how to model the proposed multiprocessor system. In section 4.1, we shall illustrate the correspondence between a multiprocessor system and a queuing network model. In section 4.2, we shall describe the input parameters and output parameters (performance measurements) for the analytical model. In this dissertation, the proposed architecture has the following two characteristics:(1) A finite buffer is provided for each switch element(link) and each memory module. (2) Each processor can only generate up to a maximum of outstanding memory requests. We, therefore, model the proposed machine as a multi-class, finite-buffer open queuing network with restricted capacity. In section 4.4, we shall present a technique to convert the proposed open queuing network into an equivalence closed queuing network. By using the closed queuing network, we can analyze the performance of the proposed multiprocessor system. A finite buffer closed queuing network may cause *blocking*. Furthermore, a *cycle blocking* queuing network can lead to deadlock. In section 4.5, we shall discuss the type of blocking and the conditions necessary for a deadlock free finite-buffer closed queuing network.

4.1 Relationship between an Open Queuing Network and the Proposed Multiprocessor System

In this section, we shall illustrate how to represent a multistage interconnection network multiprocessor system by a multi-class queuing network model, as shown in Figure 4.1 and Figure 4.2. The relationship can be depicted as following:

service centers - switch elements(links) and memory modules.

customer - packet(i.e. memory requests are generated by processors)

class of customer - each processor generates a distinct class of customer. Hence, if there have n processors, then there are n classes of customers in the network.

external arrival rate - the external arrival rate of type *i* is defined as the number of packets that are generated by processor *i* within one time unit.

arrival rate - the arrival rate of node i can be considered as the number of arrival packets from all servers within one time unit.

service rate - for a switch element, the service rate is defined as the number of packets that are transmitted from stage i to next stage i+1 within one time unit. For a memory module, the service rate equals the 1/memory access time.

routing probability - The routing probability $P_{ik,j(k+1)}$ is defined as the probability that a packet is delivered from the node i of stage k to the node j of stage (k+1). The routing probability depends on the topology of the interconnection network and the probability distribution of memory module access, which is generated by the processors.

queue limit - The queue size is the buffer size in the switches and in the memory modules.

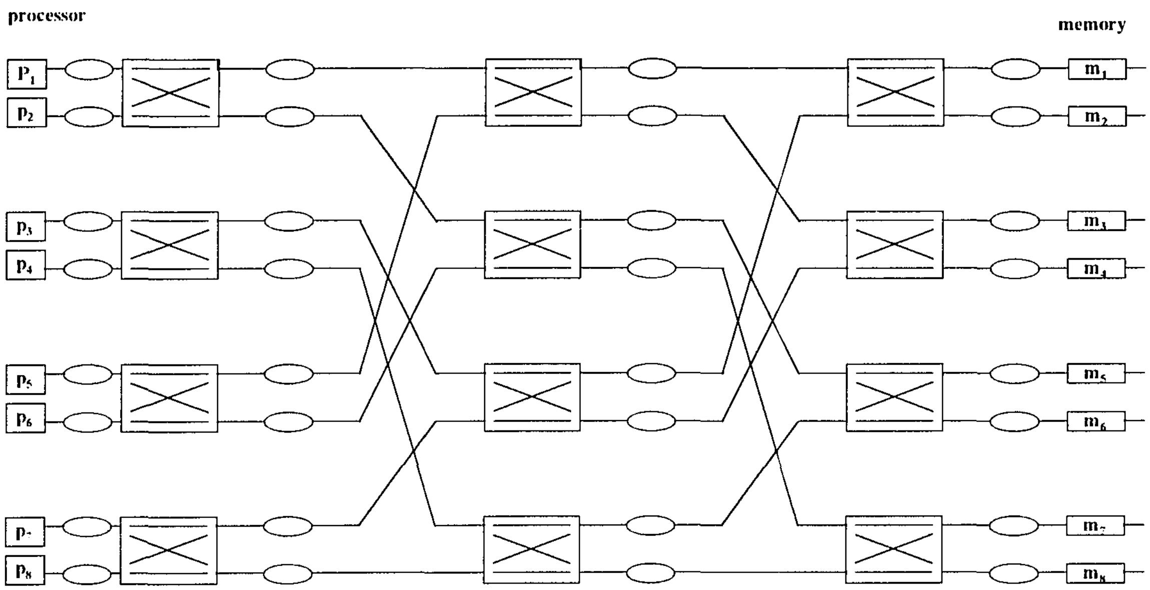


Figure 4.1 structure of a multistage interconnection network multiprocessor system

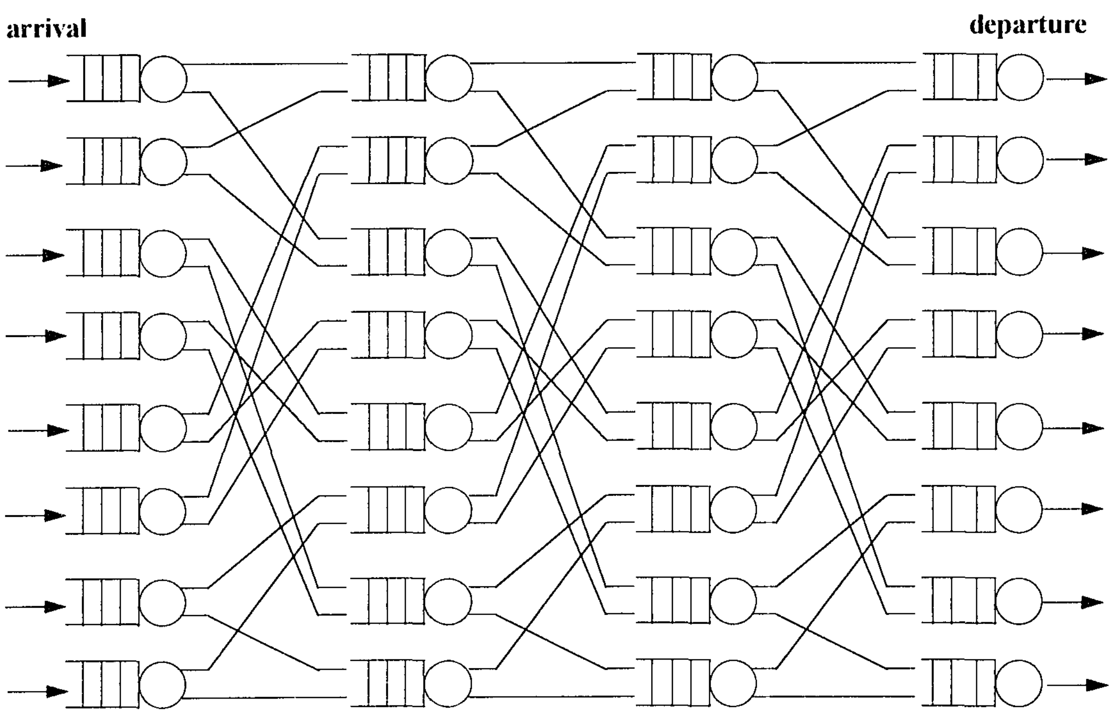


Figure 4.2 A corresponding queuing network model

4.2 Modeling the Proposed Multiprocessor System by an Open Queuing Network with Restricted Capacities

An open queuing network with restricted capacity(OQN-RC) is characterized by restricting the number of customers which can be simultaneously in the system at one time. The maximum number of customers is called the capacity of the network. In an OQN-RC system, a new customer is allowed to enter the system only if the number of customers currently in the network are smaller than the capacity, otherwise it must wait in an external queue. For the proposed multiprocessor system, each processor has a maximum number of outstanding memory requests. This implies that the number of customers of class i in the network is limited to the maximum outstanding memory requests of processor i. Therefore, the number of customers in the network can't exceed the sum of the maximum outstanding memory requests of all processors at any time point. In addition, each server queues packets by a finite-buffer in the proposed machine. Taken together, the proposed multiprocessor system can be treated as a multi-class, finite-buffer, open queuing network, with restricted capacity.

4.2.1 Input Parameters

Performance measurements for the proposed multiprocessor are obtained from the representative OQN-RC via an approximation algorithm. The analytical model requires inputs which can be classified into two categories: those that describe the hardware configuration, and those that describe the expected workload.

Hardware Configuration

The hardware configuration can be described by:

N - the number of processors.

M - the number of memory modules.

k x s switch - the size of the switches making up the forward and return interconnection networks(k-input, s-output).

network topology - a description of the topology of the interconnection network that connect processors and memory modules.

The topology of network can be described as the matrix $\Gamma=\{t_{ij}\}_{N \times \log N}$. The element $t_{ij} = [(a_{ij}, x_{a_{ij}}), (b_{ij}, x_{b_{ij}})]$ is associated with switch-$(i,j+1)$ as follows: the first ordered pair gives the address of the output pin located at the *j*th stage connected to input-0 of switch-$(i,j+1)$ and the second ordered pair gives the address of the output pin, also located at the *j*th stage, connected to input-1 of switch-$(i,j+1)$. The components a_{ij}, b_{ij} denote the switch number respectively and $x_{a_{ij}} = \{0,1\}$ denotes an output pin number at that switch. In other words,

input-0 of switch-$(i,j+1)$ - linked to output - $x_{a_{ij}}$ of switch- (a_{ij}, j)

input-1 of switch-$(i,j+1)$ - linked to output - $x_{a_{ij}}$ of switch- (b_{ij}, j)

As an example, considering an 8x8 *omega network* is shown in Figure 4-3.

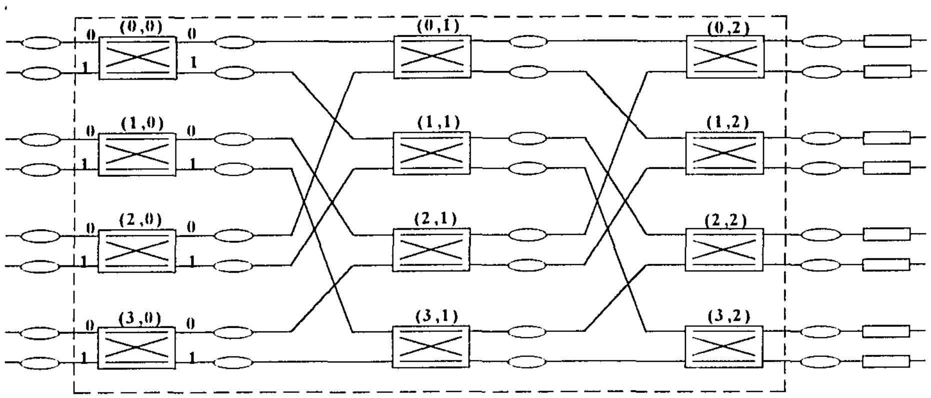


Figure 4.3 structure of an 8 x 8 omega interconnection network multiprocessor system

The topology matrix Γ of the *omega network* is given as following:

$$\Gamma = \begin{bmatrix} [(0,0),(2,0)] & [(0,0),(2,0)] \\ [(0,1),(2,1)] & [(0,1),(2,1)] \\ [(1,0),(3,0)] & [(1,0),(3,0)] \\ [(1,1),(3,1)] & [(1,1),(3,1)] \end{bmatrix}$$

Expected Workload

The expected workload in our analytical model can be described through the following parameters:

NM_i - the maximum outstanding memory requests of processor i.

λ_i - the mean inter-request rate from processor i when processor i is not blocked.

$\mu_{ij,x}$ - the mean transmission time at the output-x of the (i,j)th switch element, x=0,1.

μ_m - the mean memory access time of the mnd memory module.

$P_{ijk,l(j-1)x}$ - the probability that a packet is delivered to server $(l,j-1,x)$ from server (i,j,k), x,k=0,1.

$P_{i,j}$ - the memory referencing pattern. P_{ij} is defined as the probability that a request generated by processor i will be designated for memory module j. Obviously, $\sum_{j=1} P_{ij} = 1$

4.2.2 Output Parameters

Let K be denoted to be the sum of maximum outstanding memory requests of each processor. The performance of the proposed multiprocessor system can be measured from the following workload.

$Q_{i,j,k,x}(K)$ - the average queue length of class i customers at output-x of the (j,k)th switch element, x=0,1.

$Q_{i,j,x}(K)$ - the average queue length of customer at output-x of the (i,j)th switch element, x=0,1

$QM_{i,j}(K)$ - the average queue length of class i customers at the jth memory module.

$QM_j(K)$ - the average queue length of customers at the jth memory module.

$R_{i,j,k,x}$ - the average "residence time" (queuing plus service time) of a class I customer at output-x of the (j,k)th server, x=0,1.

$R_{i,j,x}(K)$ - the average "residence time" of a customer at output-x of the (i,j)th server, x=0,1.

$R_i(K)$ - the average response time of class i customer.

$U_{i,j,x}(K)$ - the average utilization of output-x of the (i,j)th switch element.

$UM_i(K)$- the average utilization of memory module i.

U(K) - the average utilization of the entire system.

$X_{i,j,k,x}$ (K) - the average throughput of class i customers at output-x of the (j,k)th switch element, x=0,1.

$X_i(K)$ - the throughput of class i customers.

X(K) - the throughput of the entire system.

4.3 The characteristics of the proposed model

The proposed model consists of (m+mlogm) service centers(m memory modules, mlogm links. i.e, m the number of processors) and one control node, as shown in Figure 4.4. The network is fed by m external arrival processes. The external arrival process is a *Poisson* distribution with mean rate λ_{0i}.(i.e; the mean memory request rate of processor i, without considering its blocking due to its maximum outstanding memory requests).

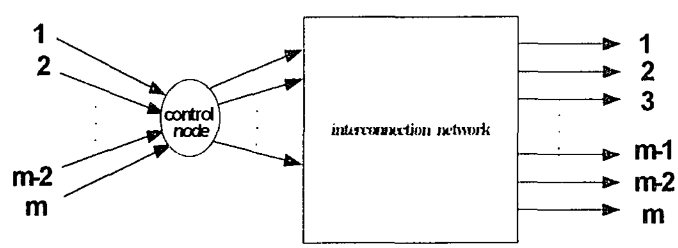
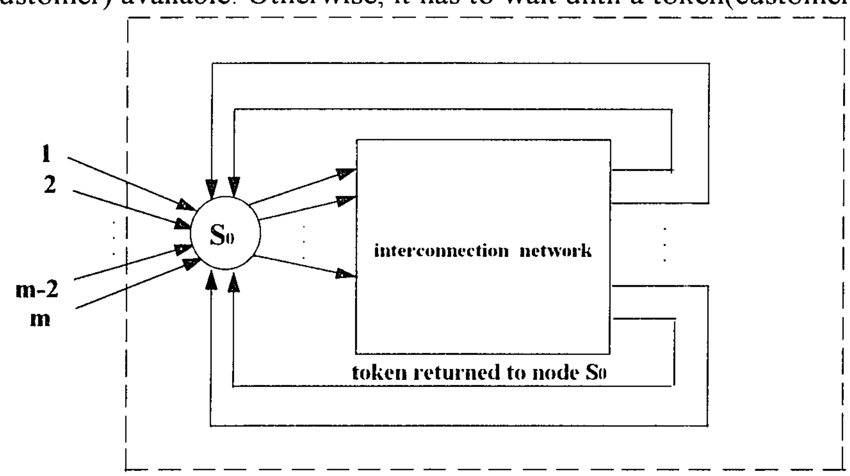


Figure 4.4. Modeling an open queuing network with restricted capacities

Each server uses a FCFS queuing discipline. The service time of each server is assumed to be exponentially distributed(if the server is not blocked). In addition, each server has a fixed finite capacity. A customer at server (i,j) is destined to server (k,j+1). If the buffer of the server (k,j+1) is not full, then the customer joins queue (k,j+1) immediately after completion service at server (i,j). Otherwise, the customer is blocked and the server (i,j) stops serving simultaneously. A customer leaves the network system after completion service at server (i,logm+1). Let C_0 be the control node. The node C_0 is used to control the number of customers entering the network. Initially, there are N classes(i.e equals the number of processors) of tokens in C_0 and the number of tokens for each class is equal to the maximum outstanding memory request for the corresponding processor. Any external customer entering the network must go through node C_0. In the proposed model, we assume that the buffer size is 1 in node C_0 for each class. The external new *i* class customer is allowed into the network immediately if there exists corresponding *i* class of tokens in node C_0. The new customer is queued in node C_0 only if there is no corresponding class token available and the corresponding type of buffer is empty. Otherwise, we discard the new customer. Once a customer is allowed into the network, the customer holds the token during its sojourn in the network. The token is released and is returned to node C_0 when it leaves the network. At the same time, the corresponding class of customer in queue of node C_0, if any, enter the system. By controlling the token flow, we know that the total number of customers simultaneously sojourning in the network can't exceed $\sum_{i=1} NM_i$ and for each type i customer can't exceed NM_i(i.e; NM_i is the number of maximum outstanding memory request for processor i).

4.4 The Technique of Converting an Open Queuing Network Model with Restricted Capacity into an Equivalent of Closed Queuing Network

In the section, we shall convert the open queuing network with restricted capacity into an equivalent of closed queuing network, as shown in Fig-4.5. The closed network consists of (m+mlogm+1) nodes: the (m+mlogm) nodes of the original network, and an extra node modeling the synchronization of the customers and the tokens. We denote S_0 to be the extra node. An equivalent view of the system is provided by *exchanging* the roles of the *customers* and the *tokens*. The tokens can be considered as the customers and the customers can be considered as the tokens, in the closed queuing network. In this case, there is a constant number of customers in the network, which is equal to $\sum_{i=1} NM_i$, the total number of tokens. From the point view, the system can be treated as a closed queuing network. In the closed queue network, a customer(token) arriving at node S_0 corresponds to a customer leaving the network in the open queuing network. Whenever a customer(token) arrives at node S_0, it is immediately served if there is the same type token(customer) available. Otherwise, it has to wait until a token(customer) arrives.



: closed queueing network (token is considered to be customer)

: the number of tokens of each type are fixed in the model

Figure 4.5 The corresponding closed queuing network

4.4.1 Proof both of Model are Equivalent

To prove two networks are equivalent, we have to show that they have the same rate matrix. These results may be proved in two steps:(1) show that both networks have the same number of states. (2) define a one-to-one mapping between the states of two networks such that the transition rates into and out of corresponding states are the same.

Proof:

case 1). to show both networks have the same number of states.

To construct a CQN from a given OQN-RC, we add a node(node S_0) to CQN. The arrival rate λ_{i0} and service rate μ_{oi} of node S_0 are defined as $\lambda_{ij0} = T(N_{ij})$ and $\mu_{ij0} = \lambda_i$, where $T(N_{ij})$ = the throughput when there are (NM_i-N_{i0}) tokens of class i in the network(i.e; N_{i0} is the tokens of class i reside in node S_0), λ_i= external arrival rate of class i. The node S_0 has the same function as the control node C_0 of OQN-RC has.

case 2). define a one-to-one mapping between the states of two networks such that transitions into and out of two equivalent states are the same.

The difference, between the open queuing network with restricted capacity and the proposed closed queuing network, is that we treat the tokens as the customers in the closed queuing network(i.e, exchange the roles of the customers and the tokens). Therefore, the routing probability of a customer throughout the closed network is exactly the same as the routing probability of the customer throughout the OQN-RC. In addition, the service rate and arrival rate of the closed network exactly equal the service rate and arrival rate of the OQN-RC in equivalent states. It is proved

4.5 Queuing Network Model with Finite Buffer

To measure the performance of queuing networks, we usually assume that the buffers in each node are infinite and analyze those models by product-form solution. This assumption is reasonable only if the mean queue lengths at all nodes are much smaller than the available buffer size in the network or under certain circumstances. For example, when $1 \leq K \leq \min\{\Psi_i\}$ K=total population in the network, Ψ_i = buffer size of node i. There is no blocking because the entire population can reside in any node. Hence in this case the network has a product form solution.

However, in the real life systems, the storage space is always finite. An important phenomena of queuing networks with finite queues is that the flow of customer through a node may be momentarily stopped when its next destination node in the network reaches its capacity. That is a phenomenon called *blocking*. Some study[1,2,7,47,53,56] have shown that blocking often occurs in real systems and is an important feature to model because in practice, *computer performance modelers are usually less concerned with accurate distribution assumptions than with accurately representing structural features of the system being modeled. This is because experience has shown that incorrect structural assumptions can cause greater modeling errors than incorrect distributional assumptions, particularly for closed queuing network*[47]. Different blocking types could cause different behavior both for job arrivals at a full node and for the nodes' activity. In section 4.5.1, we shall discuss the types of blocking

4.5.1 Type of Blocking

In a queuing network with finite buffer, a job completing the service at node i immediately goes to node j according to routing probability $p_{ij}, 1 \leq i,j \leq M$, M = the number of nodes in the network. When the populations in node j attain its maximum capacities, no other job can be enqueued. In this case, node j is said to be full, and the blocking phenomenon arises. To model different characteristics of various real life systems with finite resource, different blocking models or types must be employed to describe different behavior both for job arrivals at a full node and for the nodes' activity. In particular, each blocking mechanism defines when a node is blocked, what happens during the blocking period, and how a node becomes unblocked. In [56], blocking types are categorized into three classes.

Type I. Blocked after service(**BAS**).

A customer upon completion of its service at node i chooses to go to node j. If node j at that moment is full, the customer is forced to wait in front of server i until there is a departure from the destination node j. Server i remains blocked for this period of time and it can not serve any other customers which might be waiting in the queue. The case could happen in a **BAS** type that more than one nodes can get blocked by node j. Therefore, it is necessary to impose a priority in order to decide which node to be unblocked when there is a space available at node j.

Type II. Blocked before service(**BBS**).

A customer at node i declared its destination node j before it starts its service. If node j is full, the ith node become blocked and will become unblocked when a departure occurs from the destination node j. If the full node j blocks more than one node and then a space becomes available at node j, all blocked nodes begin their service in their respective nodes. The first customer to complete its service will move into node j and block other nodes again. When this happens, the rest of servers are suspended and become blocked. Once departure occurs from the destination node, all blocked customers are resumed from the point of interruption and receive a new service.

A **BBS** can be classified into two types, based on the blocked customer may or may not occupy the server.

Type II.1 Blocked before service - server no occupied(**BBS-SNO**)

The server is not occupied when the customer is blocked. As an example, consider a communication channel where a message is transmitted via a channel. The channel can not transmit the message when the capacities of destination node is full. Also, the channel can not hold the message.

Type II.2 blocked before service - server occupied(**BBS-SO**)

The server is occupied when the customer is blocked. As an example, a manufacturing system.

Type III. Repetitive service blocking(**RS**)

A customer completes its service at node i and attempts to join destination node j. If node j at that moment is full, the custom repeats its service at node i until a space is available at the destination node j. **RS** blocking is different from **BAS** and **BBS** blocking in that blocked node do not stop processing customers. Within this category of blocking mechanism, there are two sub-categories:

Type III.1 Repetitive service fixed destination(**RS-FD**)

Once the customer's destination is determined it can not be altered for each repeated attempt. For exponential servers, **RS-FD** blocking is equivalent to **BBS-SO** blocking because we can consider the repeat service as a wasted service. As an example, in a packet-switched network with fixed routing in which each node sends a packet and waiting for a reply. If no acknowledge after certain period, the packet is sent again.(i,e the packet that arrive the destination node and find no space available is simply lost)

Type III.2 Repetitive service random destination(**RS-RD**)

A destination node is randomly chosen at each service completion independently of the destination node chosen in the previous time. As an example, in a flexible manufacturing system where a work-piece can not be processed at a particular destination(because it is full) and has to join another station for repeat attempt.

4.5.2 Deadlock in Blocking Networks

The finite buffer of a queuing network may introduce blocking. Furthermore, in a *cycle blocking* queuing network could lead a deadlock occur. For example, a set of nodes is in a deadlock state when every node in the set is waiting for a space to become available at another node in the set. In this case, all servers in the cycle are blocked and they can not get unblocked because the space for required to change of status will never be available. In this section, the conditions of *deadlock avoidance* for the different types of blocking are discussed as following:

1). An **OQN-B** is deadlock free if there is *no cyclic* in a queuing network

2). For a blocking closed queuing network is deadlock free if and only if for each cycle, C, in the network, the following condition holds:

BAS: for each cycle C, $K < \sum_{i \in C} \Psi_i$

BBS-SNO: for each cycle C, $K < \sum_{i \in C} (\Psi_i - 1)$

BBS-SO: for each cycle C, $K < \sum_{i \in C} \Psi_i$

RS-FD: for each cycle C, $K < \sum_{i \in C} \Psi_i$

RS-RD: for each cycle C, $K < \sum_{i \in C} \Psi_i$

K : total population in the network.

Ψ_i = buffer size of node i.

In our proposed queuing network model, we restrict the population in the network. Therefore, the closed queuing network is considered to be a *deadlock free network*.

Chapter 5

Analyzing the performance of the proposed closed queuing network model

The proposed model employs a closed multi-class queuing network to represent the multiprocessor system. Performance measurements for the proposed multiprocessor system are obtained by analyzing the performance of the representative queuing network model. Currently, a product-form solution may be derived only for particular conditions and depends on the blocking types for a finite buffer system. Furthermore, an open queuing network with restricted capacity does not have a product form solution[16]. In this chapter, we shall present our approximation method ,based on the *decomposition method*, for analyzing the proposed model. The *decomposition method* decomposes a network into sub-networks, solves the sub-networks independently, and aggregates the results of those independent solutions to obtain an approximate solution to the original network. In chapter 5.1, we separate our proposed model into two subnetworks: sub1-network contains only a single node S_0, and sub2-network consists of the rest of the nodes(the interconnection network and memory modules). In section 5.2, we shall analyze the performance of the sub2-network which all buffers are assumed to be infinite(without considering any blocking phenomena). In section 5.3, we shall analyze the performance of the sub2-network by our approximation algorithm, where all buffers have finite capacities. In section 5.4, we shall aggregate the two sub-networks and obtain the performance measurements of the entire network.

5.1 Decomposing the Proposed Queuing Network into Two Sub-Networks

The proposed closed network model consists of (m+mlogm+1) nodes: $\frac{1}{2}$mlogm switch elements(i.e; a switch element is a 2x2 crossbar network), m memory modules, and an extra node S_0(modeling the synchronization of customers and tokens). The total number of customers in the network are $\sum_{i=1}^{m} NM_i$ (NM_i is the maximum number of outstanding memory requests for processor i).

5.1.1 Decomposition method

We apply the decomposition method, as this method is particularly useful when isolated features cannot be represented in product form networks. Typically, the method gives a good approximation when the network has a closed product form. The general approach of decomposition methods for queuing networks follow these steps:

1). decomposing the network into appropriate sub-network

2). solving each sub-network independently

3). combining the sub-networks solutions to form an approximate solution of the entire network.

We decompose the closed queuing network into two sub-networks: sub1-network consists of only node S_0 and sub2-network consists of all the remaining in the network.(see Fig-5.1).

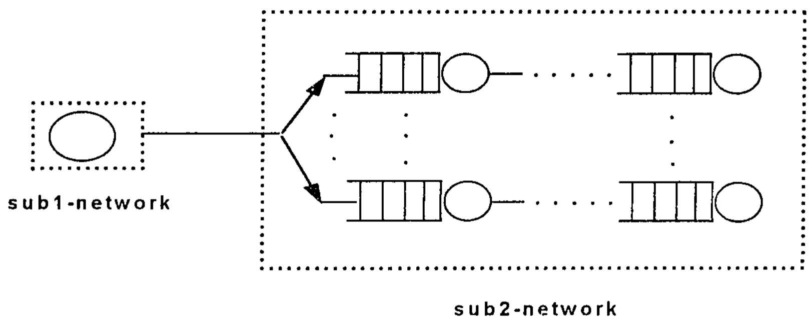


Figure 5.1 decomposing a network into two subnetworks

The Chandy-Herzog-Woo theorem states that a service center may be designed such that the remainder of the network is replaced by a single center that is "flow-equivalent" to the subnetworks it replaces. By the theorem, the proposed closed queuing network can be represented by two service centers: the designated center(sub2-network), and the remaining center(sub1-network). (see Figure 5.2)

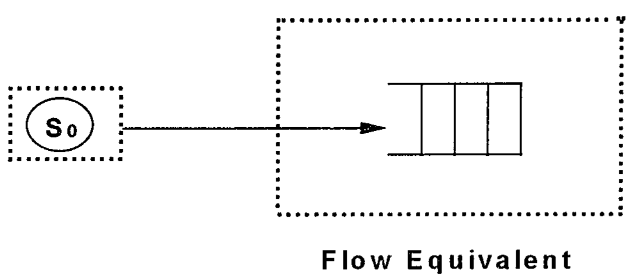


Figure 5.2 sub2-network replaced by a flow-equivalent center

5.2 Throughput Analysis for Sub2-Network

Figure 4.5 shows the throughput from the sub2-network exactly equals the arrival rate of customers entering the sub1-network. To measure the performance of the proposed model, we need to obtain the throughput $X(N_i)$, $N_i=1,2,\ldots NM_i$, of the sub2-network. The throughput $X(N_i)$ of the sub2-network depends on its characteristics(such as service rate, arrival rate of individual nodes) and the number of tokens(customers) existing in the sub2-network. In our model, the tokens are fixed and the node S_0 is used to model the synchronization of the customers and the tokens. It implies that if there are n_0 tokens in node S_0(sub1-network) then the number of tokens existing in designated center(sub2-network) is equal to the total number of tokens *minus* n_0 (tokens in node S_0). In section 5.2.1, we assume that all queues in sub2-network have infinite buffers and have a *product form solution* for the queuing network. The throughput $X(N_i)$ can be calculated by *mean value analysis* method. In section 5.2.2, where we consider a finite buffers, the network does not possess a product form solution. Thus, the throughput $X(N_i)$ shall be derived by our approximation algorithm.

5.2.1 Throughput Analysis of the Sub2-network with Infinite Buffer

In this section we shall analyze the throughput, $X(N_i)$, of the sub2-network, assuming infinite buffer capacities for all servers. The throughput, $X(N_i)$, can be computed by considering the sub2-network to be a closed queuing network with a fixed N_i customers. To compute the performance measurement of the proposed model, we shall apply the Mean Value Analysis (**MVA**) algorithm.

5.2.1.1 Mean Value Analysis

The **MVA** algorithm depends on *Little's Theorem* and *Arrival Theorem*, which state that a job in a closed queuing network upon entering a queue observes the mean state of the network with itself removed. This is intuitively appealing since we can think of the removed customer as the arriving customer himself. For a closed multi-class network, let the workload be described by the network population $\boldsymbol{K} = (K_1, K_2, \ldots, K_r, \ldots K_N)$, where K_r is the number of class r customers. We are interested in the following performance measurements:

$L_{ijr}(\boldsymbol{K})$ denotes the mean number of class r customers at node (i,j).

$W_{ijr}(\boldsymbol{K})$ denotes the mean waiting time of class r customer at node (i,j).

$Q_{ijr}(\boldsymbol{K})$ denotes the mean residence time of class r customers at node (i,j).

$T_r(\boldsymbol{K})$ denotes the throughput of class r.

$U_{ijr}(\boldsymbol{K})$ denotes the utilization of node (i,j) by class r

Applying the Arrival Theorem and Little's result, we derive the following set of recurrence equations:

$$W_{ijr}(K) = \frac{1}{\mu_{ijr}}\left(\sum_{m=1}^{N} L_{ijm}(K - 1_r) + 1\right) \tag{5.1}$$

The throughput of class r is determined from equation

$$T_r(K) = \frac{K_r}{\sum_{i=1}^{N}\sum_{j=1}^{\log N} V_{ijr} W_{ijr}(K)} \tag{5.2}$$

and the mean length of the queue at node (i,j) is given by Little's theorem.

$$L_{ijk}(k) = T_r(k) V_{ijr} W_{ijr}(k) \tag{5.3}$$

Note that $(K\text{-}1_r)$ is the network state with one less class r customer than state K.

Therefore, the performance measurements can be obtained by giving as input parameters both the fixed service rate of μ_{ijr} and the relative visit ratio V_{ijr}.

5.3 Throughput Analysis of Sub2-network with Finite Buffers

Since the actual systems have buffers with finite capacity, queuing networks with blocking must be used to model them. Since blocking causes interdependencies between stations, these queuing networks cannot be analyzed by existing product form algorithms. This implies that we must analyze a blocking queuing network by an approximation method.

5.3.1 Our Proposed Analysis Method

5.3.1.1 Interrelationship Among Switches

In the proposed finite-buffer, packet-switched interconnection network, the packets are delivered through the subsequent interconnection network stage by stage. The *omega interconnection* network, the topology of our model, is constructed from 2x2 crossbars. Each switch element has two ports connected to the previous stage The interrelationship among switches with respect to the blocking situations is explained with the following example: From figure 5.3, we see a task "A" that is being served at output-1 of switch-(i,j), and that the next destination of task "A" is at output-0 of switch-$(k,j+1)$. When the task "A" complete its service it moves to output-0 of switch-$(k,j+1)$ if there is space available. On the other hand, if when task "A" completes its service, the output-0 of switch-$(k,j+1)$ is full. the task is forced to wait at output-1 of switch-(i,j) until a position at the output-0 of switch-$(k,j+1)$ becomes available. In this situation, output-1 of switch-(i,j) becomes blocked, which implies that no service can take place at this server, i.e. the server becomes idle. Now assume that another task ,"B", located at output-0 of switch-(l,j), completes its service during this period and task "B" also is destined to output-0 of switch-(k,j+1). Consequently, output-0 of switch-(l,j) becomes blocked and the corresponding switch becomes idle. Once a position in output-0 of switch-(k,j+1) becomes available, task "A" has the priority to move to output-0 of switch-(k,j+1) since it is blocked first, task "B" joins switch-(k,j+1) when the switch again becomes available.

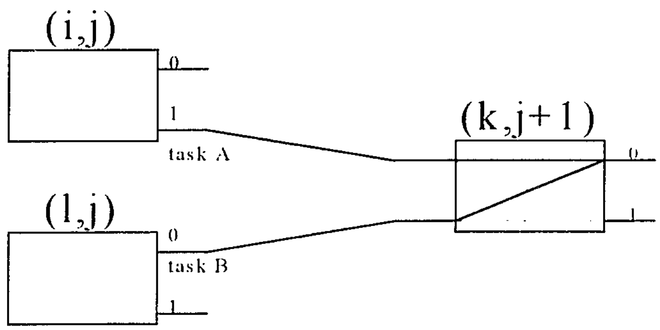


Figure 5.3 Blocking mechanism

5.3.1.2 Modeling a Single Switch

From Fig-5.3, we see that a task arriving at input-x (x=0,1) of a single switch-(*i,j*) can have either an output-0 or an output-1 destination. The process of the incoming traffic originating from both input-0 and input-1 divides itself according to *splitting* probability(which is assumed to be mutually independent) and merges at output-0 or output-1 of switch-(*i,j*). The *splitting* probability is obtained from the routing probability. We make some definitions as follows:

u_{ij0} = the probability that an arriving task is transmitted from input-0 to output-0 in switch-(i,j).

u_{ij1} = the probability that an arriving task is transmitted from input-0 to output-1 in switch-(i,j).

v_{ij0} = the probability that an arriving task is transmitted from input-1 to output-0 in switch-(i,j).

v_{ij1} = the probability that an arriving task is transmitted from input-1 to output-1 in switch-(i,j),

subject to $u_{ij0} + u_{ij1}=1$ and $v_{ij0}+v_{ij1} = 1$

The arrival process to output-0 or output-1 of switch-(i,j) is a superposition of two split processes originating at input-0 and input-1. If the two input processes are Poisson process, it follows that the resulting processes is itself Poisson, since the superposition of two or more Poisson processes is itself Poisson with rate given by the sum of the rates of the superimposed processes.

5.3.2 Approximation Algorithm

In our approximation algorithm, we shall analyze the model based on analysis of *each switch individually* stage by stage. However, we shall revise both arrival and service processes in order to take into consideration the dependency between the internal arrival processes at the input of each switch and the blocking effect described above. The analysis and the resulting algorithm we present is based on the following principles: Initially. we assume the output port of all switches of the network are treated as $(M/M/1/C_{ijx}+2)$ model. $0\leq i\leq$number of processors-1, $0\leq j\leq$number of stages-1, $0\leq x\leq 1$. Note that two units are added to the capacity of output-x of the switch-(i,j) to reflect two tasks in the servers of the previous stage that could be blocked by the output-x of switch-(i,j). The approximation algorithm has two major steps: the first step computes the *approximate service time* and the *approximate probability distribution of queue* *length* in the output ports of each stage. The iteration begins at the rightmost stage, which we assume is never to be blocked, and propagates back to the first stage. The second step works from left to right and determines the input process of each component in successive stages in terms of its service rate and its input process using *approximate* results based on the *splitting* and *superposition* of arrival processes. The input process to the first stage is assumed to be Poisson. In second step, we also compute the approximate queue-length. The two steps are then repeated until convergence is achieved, as measured by the closeness of successive approximations of the queue length which are computed by step1 and step2 respectively.

5.3.2.1 Step-1 Algorithm

An initialization step gives the first approximation to the *queue-length distribution* for the output-x (x=0,1) of each switch in the network. The approach used in this step is initially to have all outputs of the last stage (stage-(M-1)) analyzed as ($M/M/1/C_{i,M-1,x}+2$) queue, and with the algorithm proceeding *backwards* to switches of stage-0. However, unlike the last stage, at the intermediate stages, *the distribution of service time and the arrival rate at each queue have to be revised in order to reflect the effect of blocking.*

The following section we shall analyze the approximate service time distribution by considering the effect of blocking. Let $B_{ijx}(t)$ be the *effective service time distribution* at output-x of switch-(i,j) and $B^*_{ijx}(s)$ denote the *Laplace Transform*(**LST**)of $B_{ijx}(t)$, $0 \le i \le L-1$, $0 \le j \le M-1$, $x=0,1$.

The server cannot be blocked in the *last stage* of the network. The *effective service time distribution* at queue-(i,M-1,0) and queue-(i,M-1,1) are given as equations (5.4),(5.5) in term of their *Laplace Transform*.

$$B^*_{i,M-1,0}(s) = \mu^*_{i,M-1,0}(s) \tag{5.4}$$

$$B^*_{i,M-1,1}(s) = \mu^*_{i,M-1,1}(s) \tag{5.5}$$

For the *intermediate stages* of the network, the connections between a stage and its following stage are constructed as Figure 5-4. In this structure, input-0 and input-1 of switch-(*i,j*) are connected to output-1 of switch-(*k,j-1*) and output-0 of switch-(*l,j-1*), respectively. A task, after completing its service at output-1 of switch-(*k,j-1*), can find its destination(i,e;output-x of switch-(*i,j*)) in one of three states in term of the total number of tasks: *less than its capacity*, *equal to its capacity* and *equal to its capacity - 1*(i.e; another task located in output-0 of switch-(*l,j-1*) has been blocked by output-x of switch-(*i,j*) before the task requests to be serviced).

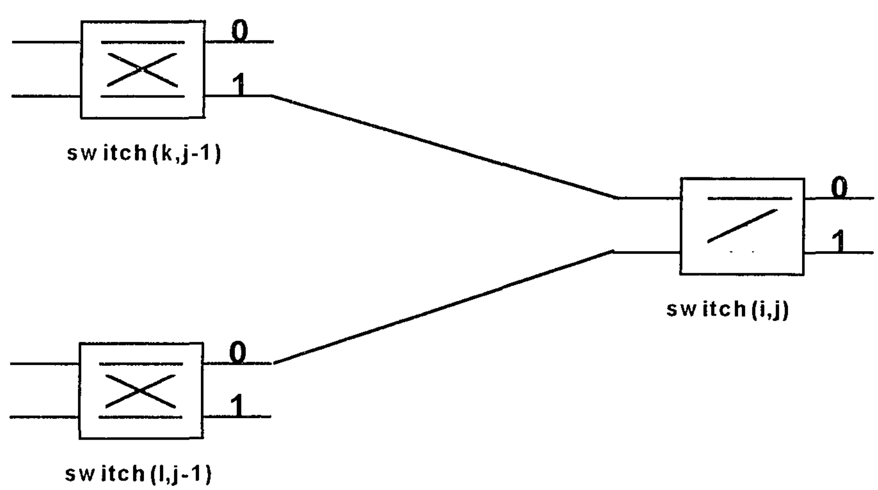


Figure 5.4 two interdeparture processes split and merge at destination

Now we discuss the *effective service time distribution* for above three states.

Case 1). The task completes service and the number of tasks located at its destination queue are less than the capacities of queue. Obviously, there is no additional delay due to blocking. It implies that the *effective service time distribution* of the task is equal to its original service time distribution.

The *effective service time distribution* can be expressed as equation (5.6)

$$B^*_{k,j-1,x}(s: s < C_{ijx}) = \mu^*_{k,j-1,x}(s), 0 \le i \le N-1, 0 \le j \le M-1 \qquad (5.6)$$

Case 2). The task completes service and the number of tasks located at its destination queue are equal to the destination's capacity. In this case, the task can not join the destination queue, and has the effect of increasing the mean residence time of the source station. Let us define that *mean remaining service time* is the time that the job in service still needs to be completed at the moment when a new job enters the system. Accordingly, the *effective service time* of the task is the sum of its *original service time* and *remaining service time* of its destination server. Let b_{ijx} denote the *remaining service time* distribution. Then *the effective service time distribution* can be expressed as equation (5.7)

$$B^*_{k,j-1,x}(s: s = C_{ijx}) = \mu^*_{k,j-1,x}(s).\frac{b_{ijx}}{s+b_{ijx}} \qquad 0 \le i \le N-1,\ 0 \le j \le M-1 \qquad (5.7)$$

The mean remaining service time can be obtained from the following theorem[3].

Theorem: The mean remaining service time of the destination station is exactly its mean service time $1/\mu$

Case 3). The task completes its service and the number of tasks located at the destination queue are the *capacities of the destination queue + 1*. This case indicates that the current task finds another task "B" located in output-0 of switch-(*l,j-1*) already blocked by switch-(*i,j*), which implies that the task "A" has to wait two service completions in order to enter its destination queue. It means that the *revised service time distribution* of the task "A" must reflect the *additional delay*. Therefore, the effective service time is the sum of its *original service time*, the *remaining service time* of its destination server, and *the service time of another task* "B". Then, the *effective service time* can be expressed as equation (5.8)

$$B^*_{k,j-1,x}(s;s=C_{ijx}+1)=\frac{b_{ijx}}{s+b_{ijx}}\cdot\mu^*_{k,j-1,x}(s)\cdot\mu^*_{l,j-1,x}(s)\quad 0\le i\le N\text{-}1, 0\le j\le M\text{-}1 \tag{5.8}$$

From equation (5.6),(5.7),(5.8), the *unconditional effective service time distribution* $B^*_{k,j-1,x}(s)$ can be derived as equation (5.9).

$$B^*_{k,j-1,x}(s)=(1-P_{C_{ijx}}-P_{C_{ijx}+1})\cdot B^*_{k,j-1,x}(s;s<C_{ijx})+P_{C_{ijx}}\cdot B^*_{k,j-1,x}(s;s=C_{ijx})+P_{C_{ijx}+1}\cdot B^*_{k,j-1,x}(s;s=C_{ijx}+1) \tag{5.9}$$

where $P_{C_{ijx}}$, $P_{C_{ijx}+1}$ are defined as:

$P_{C_{ijx}}$: The probability that when an incoming task completes its service from previous stage and there are C_{ijx} tasks queued in output-x of switch-(i,j).

$P_{C_{ijx}+1}$: The probability that when an incoming task completes its service from previous stage and there are C_{ijx}+1 tasks in output-x of switch-(i,j).

The probability $P_{C_{ijx}}$, $P_{C_{ijx}+1}$ are obtained from the (M/M/1/C_{ijx}+2) system.

Step-1 Algorithm:

```
input data();
/* last stage in the network */
for i = 0 to N-1
begin
   B*_{i,M-1,x} = μ*_{i,M-1,x}, x = 0,1
   treating and analyzing the output-x of switch-(i,M-1) by a (M/M/1/C_{i,M-1,x}+2) model;
   computing average queue-length L^(1)_{i,M-1,x} in output-x of switch-(i,M-1);
   computing average waiting time W^(1)_{i,M-1,x} in output-x of switch-(i,M-1);
   computing the probability P_{C_{i,M-1,x}} and P_{C_{i,M-1,x}+1};
end
/* Intermediate stages and first stage */
for j = M-2 down to 0
 begin
  for i = 0 to N-1
   begin
      revising the effective service time distribution B*_{ijx};
      treating and analyzing output-x of switch-(i,j) by a (M/M/1/C_{ijx}+2) model;
      computing average queue-length L^(1)_{ijx} in output-x of switch-(i,j);
      computing average waiting time W^(1)_{ijx} in output-x of switch-(i,j);
      computing the probability P_{C_{ijx}} and P_{C_{ijx}+1};
   end
 end
```

Figure 5.5 Step-1 (right to left)

5.3.2.2 Step-2 Algorithm

This step updates the approximations of the queue-length distribution and the distribution waiting time for each switch in the network by revising parameters. As in step-1, two units are added to the capacity of the output-x to indicate that two tasks in the previous step can be blocked by the output port. In this step, we also analyze each switch in isolation stage by stage. However, unlike step-1, this algorithm proceeds in the *forward direction* by determining the arrival process in consecutive stage from stage 0 to stage(M-1). In stage 0, the arrival processes are given as *Poisson process*, and the effective service time distributions are those obtained in the step-1. In the intermediate stages, we shall compensate for the blocking effect by revising the *interarrival time* and the *interdeparture time*. All servers are analyzed as an (M/M/1/C_{ijx}+2) system. The *revised interdeparture* and *interarrival process* associated with output-x of switch-(i,j), $0 \le i \le N-1, 0 \le j \le M-1$, are defined as following:

$D^*_{ijx}(S)$: the LST of the distribution of the *effective interdeparture time* from output-x of switch-(i,j), x=0,1.

$A^*_{ijx}(s)$: the LST of the distribution of the *effective interarrival time* to input-x of switch-(i,j), x=0,1.

Departure Processes

In this section, we shall derive the LST of the distribution of the interdeparture time from output-x of switch-(i,j). In order to obtain the LST of the distribution of the *effective interdeparture process* from output-x of switch-(i,j), we need to know the LST of the distribution of the interarrival process to the output-x of switch-(i,j) and the service time at the output-x. We assume that service time at these outputs are independent. A task complete service and departure from its service center, if at least one task is left behind in the queue, the first task in the queue comes into service and the next departure may occur at the end of its service. Otherwise, if the task leaves behind queue empty then the next departure occurs *after an arrival and its subsequent service time*. Let π_{ijx} be the probability that a departing task leaves queue-x of switch-(i,j) empty. The probability π_{ijx} can be derived from (M/M/1/C_{ijx}+2) system by the given arrival rate and service rate, which are obtained from step-1. Therefore, the LST of the departure process from output-x of switch-(i,j), $D^*_{ijx}(s)$, can be formulated as equation(5.10):

$$D^*_{ijx} = \pi_{ijx}\theta^*_{ijx} \cdot B^*_{ijx}(s) + (1-\pi_{ijx}) \cdot B^*_{ijx}(s) \tag{5.10}$$

Where $\theta^*_{ijx}(s)$ is the LST of the distribution of the next arrival time after a customer complete its service.

Arrival Processes

In a multistage interconnection network we consider an arbitrary switch, say switch-(*i*,*j*), whose input-0 and input-1 are connected to output-1 of switch-(*k*,*j*-*1*) and output-0 of switch-(*l*,*j*-*1*), respectively.(see Fig-5.4). It follows that the *effective interarrival process* of the output-x of switch-(*i*,*j*) can be obtained from the interdeparture process of the output-1 of switch-(*k*,*j*-*1*) and the interdeparture process of the output-0 of switch-(*l*,*j*-*1*). The two interdeparture processes *split* by independent probabilities (depending on the memory reference pattern and the network topology) and *merge* in output-x of switch-(*i*,*j*) to form the approximate effective interarrival process of output-x of switch-(*i,j*). We now proceed to compute the *approximate effective interarrival process* to each output of all switches in the network.

Splitting. The "splitting" method states that if a stream with rate λ and squared coefficient of variation C is split into k streams, with each being selected independently according to probabilities P_i, i= 1,2 ....K.(Figure 5.6), then the results of each split process is given, in terms of mean and squared coefficient

of variable, as following equations (5.11),(5.12)

$$\lambda_i = \lambda P_i \qquad (5.11)$$

$$C_i^2 = P_i C^2 + 1 - P_i \qquad (5.12)$$

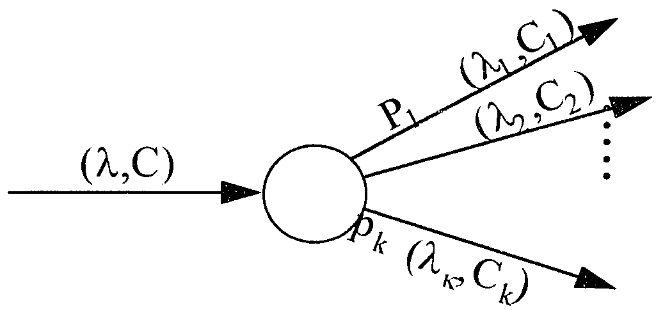


Figure 5.6 splitting processes

Let A^*_{ij0x} be the arrival process from the output-1 of switch-(*k,j-1*) to the output-*x* of switch-(*i,j*), and A^*_{ij1x} be the arrival process from the output-0 of switch-(*l,j-1*) to the output-*x* of switch-(*i,j*). By the *splitting* method the arrival process A^*_{ij01} can be obtained from the interdeparture process $D^*_{k,j-1,1}$ (from output-1 of switch-(k,j-1)) times the probability u_{ij0} (the probability an arriving task has to be transmitted from input-0 to output-0 in switch-(i,j)). The arrival process A^*_{ij01} can be obtained from the interdeparture process $D^*_{k,j-1,1}$ times the probability u_{ij1} (the probability an arriving task has to be transmitted from input-0 to output-1 in switch-(i,j)). The arrival process A^*_{ij10} can be obtained from the interdeparture process $D^*_{l,j-1,0}$ (from output-0 of switch-(l,j-1)) times the probability v_{ij0} (the probability an arriving task has to be transmitted from input-1 to output-0 in switch-(i,j)). The arrival process A^*_{ij11} can be obtained from the interdeparture process $D^*_{l,j-1,0}$ (from output-0 of switch-(l,j-1)) times the probability v_{ij1} (the probability an arriving task has to be transmitted from input-1 to output-1 in switch-(i,j)). In summary, we express those interarrival processes as equations (5.13),(5.14),(5.15),(5.16)

$$A^*_{ij00} = D^*_{k,j-1,1} \cdot \mu_{ij0} \qquad (5.13)$$

$$A^*_{ij01} = D^*_{k,j-1,1} \cdot u_{ij1} \qquad (5.14)$$

$$A^*_{ij10} = D^*_{l,j-1,0} \cdot v_{ij0} \qquad (5.15)$$

$$A^*_{ij11} = D^*_{l,j-1,0} \cdot v_{ij1} \qquad (5.16)$$

Superposition. We apply the technique of W. Whitt[65], which gives an approximation renewal process of the superposition of two or more incoming processes(see Figure 5.7).

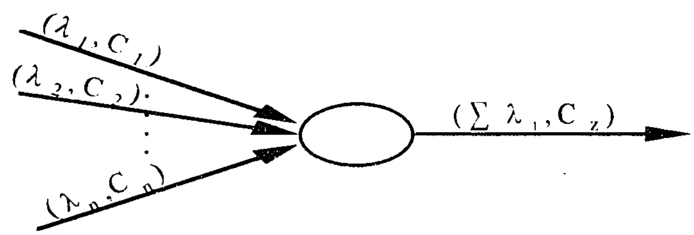


Figure 5.7 superposition of n processes

The approximation technique is based on the methods: the *asymptotic method* and the *stationary internal method*. By the *asymptotic method*, the superposition squared coefficient of variation C_{A}^{2} as a function of component squared coefficients of variation C_{i}^{2} and the arrival rates λ_i is given by:

$$C_{A}^{2} = \sum_{i=1}^{m} (\lambda_i / \sum_{k=1}^{m} \lambda_k) C_{i}^{2} \qquad (5.17)$$

The convex combination of the *asymptotic method* and the *stationary-interval method* can be determined as follows:

$$C_{H}^{2} = W C_{A}^{2} + (1-W) C_{SI}^{2} \qquad (5.18)$$

where W is a function of ρ and the rates v. The weighting function W is suggested by simulation method[65]

$$W = [1 + 2.1(1-\rho)^{1.8} v]^{-1} \qquad (5.19)$$

where

$$\rho = \sum_{i=1}^{m} \lambda_i / \mu$$

$$v = [\sum_{i=1}^{m} (\lambda_i / \sum_{k=1}^{m} \lambda_k)^2]^{-1}$$

For our model, we consider the case of two components. Let λ_X be the arrival rate and C_X^2 be the coefficient of variation of X process. Similarly λ_Y and C_Y^2 are the parameters of process Y. The parameters λ_H and C_H^2 of the superposed process, are determined as follows:

$$\lambda_H = \lambda_X + \lambda_Y \tag{5.20}$$

$$C_H^2 = W\left(\frac{\lambda_X C_X^2 + \lambda_Y C_Y^2}{\lambda_X + \lambda_Y}\right) + (1 - W) \tag{5.21}$$

Consequently, the interarrival process at output-x of node-(i,j), A_{ijx}^*, can be obtained by the superposed process of A_{ij0x}^* and A_{ij1x}^*. This is expressed as follows:

$$A_{ijx}^* = A_{ij0x}^* + A_{ij1x}^* \tag{5.22}$$

Again, like the step-1, we treat the system as (M/M/1/C_{ijx}+2) model. The performance measurements (average queuing length and average waiting time in each output-x of switches) can be computed by using the updated parameters. The step-2 algorithm is summarized in Figure 5.8.

Step-2 Algorithm:

```
/***************************************************************************/
/* k, l are the kth and the lth switches in the ( j-1) stage which are connect to switch-(i,j)  */
/* w,z,x =0,1                                                                                   */
/* M: the number of processors                                                                  */
/* L : the number of stages                                                                     */
/***************************************************************************/
```

for j = 1 to M-1

begin

for i = 1 to L-1

begin

Computing the effective departure rate $D^*_{k,j-1,w}$ and $D^*_{l,j-1,z}$;

Obtaining A^*_{ij0x} and A^*_{ij1x} (Splitting);

Obtaining A^*_{ijx} (Superposition);

Analyzing output-x of switch-(i,j) by (M/M/$C_{i,j}$+2) model.

Computing average queue-length $L^{(2)}_{ijx}$ in output-x of switch-(i,j);

Computing average waiting time $W^{(2)}_{ijx}$ in output-x of switch-(i,j);

Computing the probability $P_{C_{ijx}}$ and $P_{C_{ijx}+1}$;

end

end

Figure 5.8 Step-2 algorithm

Step-2 produces the approximate average queue-length distribution in each output of all switches, and result is compared to the average queue-length distribution, which is obtained in step-1. If the difference of the compared result is larger than the prescribed value. The effective interdeparture time distribution obtained in step-2 replaces the service time distribution used in previous step-1 and the computation returns to step-1. The step-1 and step-2 are repeated until convergence is achieved. When the repeat-loop finishes, we finally obtain the approximate queue length distribution, approximate average waiting time and throughput at each output port and the entire system.

Algorithm(step-1 and step-2)

/* NM_i : the number of maximum outstanding memory request of processor i*/

Reading data();

for i= 1 to NM_i

begin

  repeat

    step-1;

    step-2;

  until all $\left| L_{ijx}^{(1)} - L_{ijx}^{(2)} \right| \in$

  computing the approximation response time;

  computing the approximation throughput;

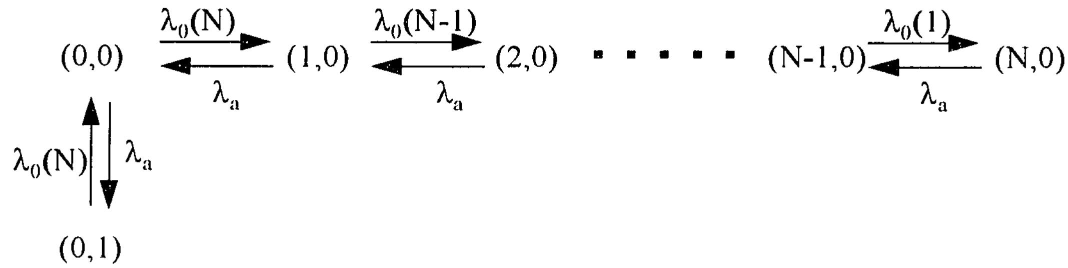
end

where $L_{ijx}^{(i)}$ be the queue length in the output-x of switch-(i,j) in the step-i.

Figure 5.9 Algorithm

5.4 Aggregating the two Sub-Networks

The proposed closed queuing network consists of (m+mlogm+1) service centers and $\sum_{i=1}^{m} NM_i$ customers(tokens). We analyze the performance of the closed queuing network by decomposing the network into the sub1-network(node S_0) and the sub2-network. In section 5.3, we compute the approximate throughput, $X(n_i)$, of the sub2-network by our proposed algorithm, where n_i is the number of the ith class tokens residencing in the network, $0 \le n_i \le NM_i$.. In this section, we shall apply our approximate throughput to obtain the average number of tokens in the sub2-network. From the Figure-4.5, we see that the throughput of the sub2-network equals the arrival rate of the sub1-network. In order to estimate the approximate average number of tokens in the sub2-network, we calculate the average number of tokens residence in S_0. The average number of tokens in S_0 can be computed from the following algorithm: Let the state of node S_0 is defined as $\mathbf{n}_0 = (n_0, n_a)$, where n_0 is the number of customers (tokens) currently present, and n_a is the number of external resources(packets). Let $\lambda_0(n_0)$ be the rate that customers are returned to S_0(i.e; the throughput of sub2-network when n_0 tokens exist in the sub2-network), and λ_a be the rate that the tokens(packets) enter node S_0. In our model, node S_0 is used as a control node. Therefore, the only stable states reached by the system are any $\mathbf{n}_0 = (n_0, 0)$ $n_0 = 0,..NM_i$ and $\mathbf{n}_0 = (0, n_a)$ $n_a = 0$ or 1. The reason is that customers(tokens) leave node S_0 immediately when external resources(packets) are available. The state transition diagram in S_0 can be described by Figure 5.10. In Figure 5.10, we set $N = NM_i$



n_0 : number of tokens

n_a : number of customers

Figure 5.10 state transition diagram for node S_0 in state $\mathbf{n_0}(n_0,n_a)$

We denote the steady-state probability of state $\mathbf{n}_0$ as $P_0(\mathbf{n}_0)$. Then the probability of state $\mathbf{n}_0$ can be determined by the following balance equations:

$P(0,1)\lambda_0(N)+P(1,0)\lambda_a=P(0,0)\lambda_0(N)+P(0,0)\lambda_a$

$P(0,0)\lambda_0(N)+P(2,0)\lambda_a=P(1,0)\lambda_0(N-1)+P(1,0)\lambda_a$

$P(1,0)\lambda_0(N-1)+P(3,0)\lambda_a=P(2,0)\lambda_0(N-2)+P(2,0)\lambda_a$

$P(2,0)\lambda_0(N-2)+P(4,0)\lambda_a=P(3,0)\lambda_0(N-3)+P(3,0)\lambda_a$

•

•

•

$P(N-2,0)\lambda_0(2)+P(N,0)\lambda_a=P(N-1,0)\lambda_0(1)+P(N-1,0)\lambda_a$

$$P(N-1,0)\lambda_0(1) \qquad = P(N,0)\lambda_a \tag{5.23}$$

In general, the equation can be expressed as

$$P(n_0-1,0)\lambda_0(N-n_0+1)+P(n_0+1,0)\lambda_a=P(n_0,0)\lambda_0(N-n_0)+P(n_0,0)\lambda_a \quad n_0=1,\dots,N-1 \tag{5.24}$$

Summing all equations in (5.24), we obtain

$$P(0,0)\lambda_a=P(0,1)\lambda_0(N) \tag{5.25}$$

$$P(0,1)=P(0,0)\frac{\lambda_a}{\lambda_0(N)} \tag{5.26}$$

From equation (5.23), we know

$$P(N,0)=P(N-1,0)\frac{\lambda_0(1)}{\lambda_a} \tag{5.27}$$

Substituting equation (5.27) into equation (5.23), we obtain

$$P(N-2,0)\lambda_0(2)=P(N-1,0)\lambda_a \tag{5.28}$$

which implies

$$P(N-1,0)=P(N-2,0)\frac{\lambda_0(2)}{\lambda_a} \tag{5.29}$$

From equation (5.29), we know

$$P(N,0)=P(N-2,0)\frac{\lambda_0(1)\lambda_0(2)}{\lambda_a^2} \tag{5.30}$$

.

.

.

Similarly, we can derive

$$P(N,0)=P(0,0)\frac{\lambda_0(1)\lambda_0(2)\dots\lambda_0(N)}{\lambda_a^N} \tag{5.31}$$

Generally, the probability of state $\mathbf{n}_0=(n_0,0)$ can be expressed as

$$P(n_0,0)=P(0,0)\frac{\lambda_0(1)\lambda_0(2)\dots\lambda_0(n_0)}{\lambda_a^{n_0}} \quad \text{,where } n_0=1,2,\dots N. \tag{5.32}$$

In the system, the number of states $\mathbf{n}_0=(n_0,0)$ are finite. Therefore, the probability state $P(n_0,0)$ can be derived from following equations

$$P(0,1)+P(0,0)+P(1,0)+\dots\dots\dots+P(N,0)=1 \tag{5.33}$$

Now, we substitute the probability of state $P(n_0,0)$, $n_0=1,2\dots N$, and the probability of $P(0,1)$ in terms of $P(0,0)$. We obtain

$$P(0,0)\frac{\lambda_a}{\lambda_0(N)}+P(0,0)+P(0,0)\frac{\lambda_0(N)}{\lambda_a}+P(0,0)\frac{\lambda_0(N)\lambda_0(N-1)}{(\lambda_a)^2}+\dots+P(0,0)\frac{\lambda_0(N)\lambda_0(N-1)\dots\lambda_0(2)\lambda_0(1)}{\lambda_a^N}=1 \tag{5.34}$$

$$P(0,0)[\frac{\lambda_a}{\lambda_0(N)}+1+\frac{\lambda_0(N)}{\lambda_a}+\frac{\lambda_0(N)\lambda_0(N-1)}{(\lambda_a)^2}+\dots+\frac{\lambda_0(N)\lambda_0(N-1)\dots\lambda_0(2)\lambda_0(1)}{\lambda_a^N}]=1 \tag{5.35}$$

Therefore

$$P(0,0)=\frac{1}{\frac{\lambda_a}{\lambda_0(N)}+1+\frac{\lambda_0(N)}{\lambda_a}+\frac{\lambda_0(N)\lambda_0(N-1)}{(\lambda_a)^2}+\dots+\frac{\lambda_0(N)\lambda_0(N-1)\dots\lambda_0(2)\lambda_0(1)}{\lambda_a^N}} \tag{5.36}$$

5.5 Calculating the Performance Measurements

5.5.1 Obtaining the Average Number of Tokens in Sub1-Network

In this section, we shall derive the average number of tokens residence in the node S_0. Let $P_0(n_0)$ be denoted as the probability of there being tokens n_0 in the node S_0. Let NT_0 denote the average number of tokens residing in the node S_0. Let NT be denoted as the average number of tokens residence in sub2-network. Then

$$P_0(n_0) = P(n_0,0) \quad n_0=1,2\dots N \tag{5.37}$$

$$\begin{aligned} P_0(0) &= P(0,0)+P(0,1) \\ &= P(0,0)+P(0,0)\frac{\lambda_a}{\lambda_0(N)} \\ &= P(0,0)[1+\frac{\lambda_a}{\lambda_0(N)}] \end{aligned} \tag{5.38}$$

$$NT_0 = \sum_{n_0=1}^{N} n_0 P_0(n_0) \tag{5.39}$$

From equation (5.39), the average number of tokens residence in the sub2-network can be obtained as following.

$NT = NM\text{-}NT_0$

5.5.2 Obtaining the Performance Measurements of the Overall System

In section 5.5.1, we obtained the approximate average number of tokens(customers) in the sub2-network. Substituting the average tokens in the sub2-network in our proposed approximate algorithm, we can derive the approximate overall performance of our system

Chapter 6
Simulation

In chapter 5, we presented an approximation algorithm to evaluate the performance of the proposed multiprocessor system. For the validation of the accuracy of our proposed algorithm, we have written a simulation program to verify these results. The simulation program is written in Borland C^{++} language. The simulation program can be used to evaluate the performance for both *blocking before service type* and without blocking network. In section 6.1, we shall describe in details the components and organization of the simulation program, including the input parameters, output parameters, the function of each components and the algorithm of each function. In section 6.2, we shall obtain the results of simulation by considering different input parameters and compare the result of simulation to the result of our analytical method.

6.1 Simulation Program

6.1.1 Components and Organization of the Simulation Program

In this section, we describe the subroutines which make up our simulation program. The subroutines are categorized into the following components:

main program: A subprogram called main() invokes the initialization routines, and the subroutines of the simulation program. The main program also check for termination and invoke the report generator when the simulation is over.

System state: The function of *system state* is used to collect state variables to describe the system at a particular time. Those subroutines include clock().

Event list routine: These programs are used to change the status of event list. Those programs include insert_arrive(), insert_depart(), move_arrive(), remove_depart(), insert_q(), insert_q0(), insert_s(), remove_q(), remove_q0, remove_s.

Statistical counters: Variables used to store the statistical information about system performance. Those variables are updated in subprogram update_time_avg_stats().

Initialization routine: The subprogram is used to initialize the simulation model at time zero or simulation completion for each run.

Timing routine: The subprogram is used to determine the next event from the event list and then advance the simulation clock to the time when that event is occur. The timing routine include timing().

Event routine: The subprogram is used to update the system state when a particular type of event occurs. Those event routines include arrive(), complete(), departure(), departure_or_complete(), dispose().

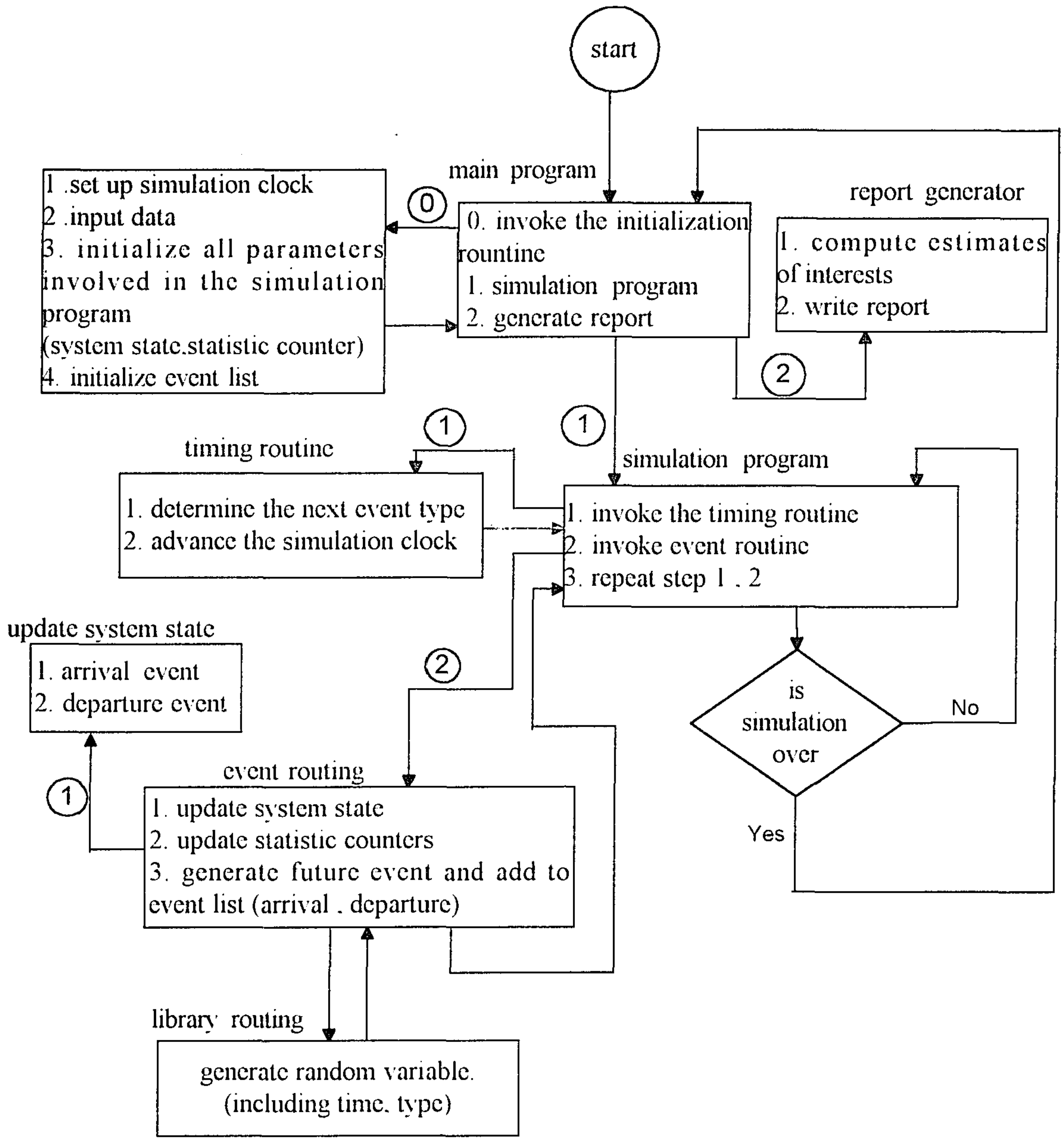
Library routines: The subprogram is used to generate random observation from probability distributions. The library routine includes rand_num(), next_arrive_expon(), next_depart_expon().

report generator: The subprogram is used to compute estimates of the desired measures of performance and produce a report when the simulation end. The subprograms include statistic(), report().

6.1.2 Flow Chart of the Simulation Program

In this section, we shall present the logical relationship among those components in our program(see Figure 6.1). The simulation program begins at time 0 with the main program invoking the initialization routine where the simulation clock, the system state, the statistical counters, and the events lists are initialized. After control has been returned to the main program, it calls the subprogram(simulation()). Then the simulation program invokes the timing routine to determine which type of event is most imminent(arrive or departure). Then the simulation clock is advanced to the time that the most imminent event will occur and control is returned to the simulation() subprogram. The simulation()

subprogram invokes event routine, which logical relationship among the activities is shown in Figure 6.2. The major activities in the event routine include: (1). the system state(arrival list or departure list) is updated to account the fact that an event type *i* has occurred; (2). to check the system available resources(tokens, buffers, server) in order to determine the next step; (3). the times of occurrence of future events are generated and this information is added to the event list; (4) information about system performance is gathered by updating the statistical counters. After all the processings have been completed, a check is made to determine if the simulation should be terminated. If it is time to terminate the simulation program, then the report generator is invoked to compute the estimates of the desired measures of performance and to produce a report. If it is not yet time to terminate, then the timing routine-event routine-termination check cycle is repeated until the stopping condition is eventually reached.



6.1 Flowchart for simulation program

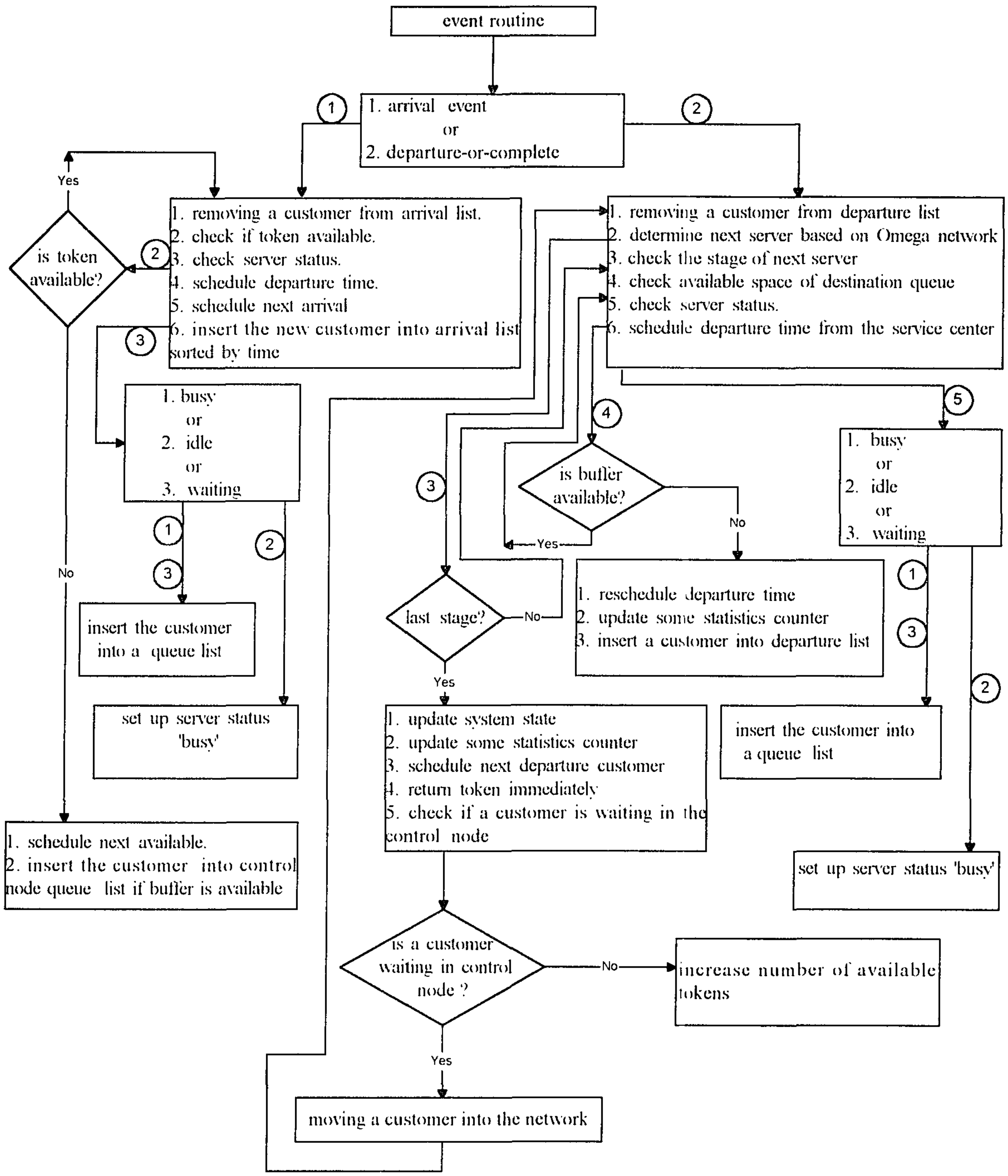


Figure 6.2 Flowchart of event routine

6.1.3 **Subroutines**, **Functions**, and **TURBO C++** variables for the queuing network model

Subprogram	**Purpose**
ARRIVE	Event routing to process arriving customer
COMPLETE	Customer competes service in the network and returns token immediately
DEPARTURE	Event routing to process departure customer
DEPARTURE_OR _COMPLETE	To decide a customer's advancing to the next stage or completing service in the queuing network
DISPOSE	Free dynamic pointer
GEN_MESSAGE _LENGTH	To generate packet length
INPUT	To read parameters data
INSERT_ARRIVE	To insert an arriver into arrive_list event
INSERT_DEPART	To insert an arriver into departure_list event
INSERT_Q	To insert an arriver into the tail of queue[row,column]
INSERT_Q0	To insert an arriver into the tail of Q0 queue
INSERT_S	To insert an arriver into the tail of server[row,column]
RAND_NUM	To generate a random number
REMOVE_DEPART	To remove a customer from departure_list event
*REMOVE_Q	To remove a customer from head of queue[row,column]
*REMOVE_Q0	To remove a customer from head of Q0[row]
*REMOVE_S	To remove a customer from server[row,column]

REPORT	To generate report when simulation ends
RUN_COMPUTE	To update statistic variable each run
RUN_INITIALIZE	To initialize variables for each run
SIMULATION	To run the simulation function
STATISTIC	To compute statistics for the variables when simulation ends
TIMING	To determine the event type for next-event
UPDATE_TIME_AVG_STATS	To update continuous_time area_accumulator statistics just before each event occurrence

6.1.4 Algorithm

In the section, we give the detail algorithm for some major subroutines as following:

```
void main(void)
{
 open input,output file;
 read information data;
 initialize all statistic variables;
 repeat simulation until repetition numbers are reached;
 compute overall statistic variables;
 generate report;
 close file;
 end simulation.
}

void arrive(void)  /* arrival event function */
{
  initial forward flag to be FALSE;
  release a customer from arrival list event;
  if the new customer can be moved to stage 1 immediately
   {
    decrease token number;
    remove the customer from queue 0 list;
    insert the customer to stage 1 queue list;
    set up forward flag to be TRUE ;
   }
  if forward is TRUE and server is idle
   {
    remove the customer from stage 1 queue list;
    schedule departure time for the customer;
    insert the customer into service list;
    insert the customer into departure list;
   }
 increase the number of arrival customers:
 generate a random number of packet length for a new customer;
 generate a new customer;
 all informations are given for the new customer;
 insert the customer into the arrival list by arrival time;
 if (token is not available) or (buffer of stage 1 is not available)
    {
    insert new customer into queue 0 list
    }
 else
    {
     if begin to statistic variable
          {
          incease the number of customers that depart from queue 0 list
          }
      decrease the number of token;
      if server of stage 1 is not idle
         {
```

```
            insert the customer into queue list
            }
        else
           {
            schedule departure time for the customer;
            insert the customer into service list;
            insert the customer into departure list
            }
      }
}


void complete();  /* customer completes event function */
{
  remove a customer from the server of last stage;
  increase the number of completion customers;
  return token(s) immediately;
  if begin to statistic variable
   {
    update statistic parameters;
    }
  dispose the customer;
 }


void departure();   /* departure event function */
{
 remove a customer from server;
 if destination server is not idle
  {
   insert the customer into destination queue
   }
 else
  {
   the customer is served immediately;
   schedule departure time of the customer from the server;
   if begin to statistic variable
    {
     number of departure customer from the server is increased by 1
     }
   insert the customer into destination server;
   insert the customer into destination server departure list;
   }
 if departure customer is from queue 0(control node)
   {
     while  (the same type of customer in queue 0 can be moved into network)
     {
      decreased the type of available token by the size of the customer;
      remove customer from queue 0 (control node);
      insert the customer to destination queue;
      repeat the while loop
     }
 if the queue of the departure customer is not empty
```

```
    {
     remove a customer from the queue;
     schedule the departure time for the customer;
     insert the customer into destination server list;
     insert the customer into departure list;
    }
}


void departure_or_complete()  /* the function handles customer advancing to next stage
                                 or completes service in the queuing network */
 {
  remove a customer from departure list;
  column number plus 1;
  search row position for next column;
  if next column is not the last stage
   {
    if the buffer of next destination node is not full and has enough space
      {
       call departure function (advance to next server)
      }
    else
      {
       re-schedule the departure time(the customer is blocked);
       update statistic parameters;
       change the server status to be waiting;
       set up the time for server to change waiting status;
       insert the customer into departure list;
      }
    }
   else
    {
     call complete function;
    }
  }


void insert_arrive(); /* insert an arrival into arrival list , customer is sorted by time */
 {
  create a new customer for arrival list;
  insert the customer into arrive_list in the order of arrival time;
  increase the number of customer in the arrival_list;
 }


void insert_depart();
{
 create a new customer for departure list;
 insert the customer into departure-list in the order of departure time;
 increase the number of customer in the departure list;
}
```

```
void insert_q();
{
 insert a customer into the queue_list of next server;
 the customer is inserted in the tail of queue list(FCFS);
 decrease the buffer size of the queue;
 increase the number of customer in the queue;
}


void insert_q0();
{
 insert a customer into the tail of queue 0 list;
 increase the number of customer in the qo list;
 decrease the buffer size of the queue;
}

void inser_s();
{
insert a customer to the server;
set up the customer arrival time;
set up the server to be busy status;
set up the customer departure time from the server;
}

void timing() /* The function is used to determined the event type for next event to
occur*/
{
 set up next_event_type;
 if arrival list is not empty
 {
   next_event_type = arrive;
   set up next event time ;
  if (departure list is not empty ) and (next departure time <  next arrival time)
      {
       next_event_type = departure;
       set up next event time;
      }
  }
 else
  {
   if departure list is not empty
    {
     next_event_type = departure;
     set up next_event_time;
    }
  }
if (arrival list and departure list are empty)
   {
    printf("error message")
    stop simulation
   }
```

```
advance the simulation clock;

void update_time_avg_stats() /* update area accumulators for statistics  purpose*/
{
 compute time interval since last event;
 update last_event_time maker;
 if (start to statistic }
  {
   update queue length area;
   update token utilization area;
   update server utilization area;
  }
}
```

Chapter 7

Numerical Results:Verification and Discussion

In chapter 5, an approximation algorithm is developed and presented to measure the performance of the proposed multiprocessor system. To demonstrate the accuracy of our proposed algorithm, we have verified it by using a simulation study and we present the simulation model and program in chapter 6. In this chapter, section 7.1, we shall compare the numerical results computed by our algorithm to those from the simulations. In section 7.2, we shall discuss the numerical results obtained and try to explain their significance and meanings.

7.1 Comparison of Numerical Results and the Simulation Results

In this section, we present a set of numerical examples in order to test the accuracy of the approximation algorithm proposed in this dissertation. We shall compare the numerical results of our algorithm to those of our simulation. We examine a 4 x 3 network(2 stages switch elements, 1 stage memory module and each stage has 4 output ports). Stage to stage is connect by the topology of shuffle exchange. We executed 108 examples with different structures and with different parameter values. These examples are classified according to the buffer size. The buffer sizes of the same stage are identical. We define S={(i,j,k) | i = the buffer size of stage 1, j= the buffer size of stage 2, k= the buffer size of memory module}. For buffer size=(1,1,5), we study the performance results by considering token numbers 4,7 and 10, respectively, in each setting. For buffer size=(1,1,8), we study the performance results by considering token numbers 8,12 and 16, respectively, in each setting. For buffer size=(5,5,5), we study the performance results by considering token numbers 10,15 and 20, respectively, in each setting. For the parameters of measuring the performance of the model, we assume the average external arrival time is 1 and the average service time ranges from 0.7 to 0.95 in steps of 0.05. The results of the performance measurements obtained by our approximation algorithm are summarized in tables 1-9. Each table contains the performance measurements for the same structure type with a different service rate. We also conducted 20 runs for each example of our simulation program, each run terminated after 2000 customers completed services in the network. Various statistics such as the average waiting time, average queue length, server utilization and standard deviations are obtained from the simulation. The numerical results of our algorithm to that of our simulation for the same structure under different parameters are summarized in tables 10-12.

7.2 Discussion

One of the major problems encountered in a multiprocessor system is that as the number of processors grow beyond a certain point the effective system bandwidth drops off due to resource access conflicts. Several studies[9,11] that have measured the performance of several existing supercomputer systems report performance degradation of over 50% due to the high number of processors. These are substantial losses for a very expensive system. In our research, we give an expanded study on how the machine's performance depends on: the characteristics of processors, the number of processors, the topology and the characteristics of the interconnection network, the characteristics of memory module and the number of memory modules. We develop and present an approximation algorithm to measure the performance of a multistage multiprocessor system(in chapter 5). We also derive formulas to obtain the scale relationship among processors, interconnection network and memory module.

A simulation study was used to demonstrate the accuracy of our developed approximate algorithm. Several performance measurements for the multi-stage multiprocessor system under study for the cases from moderate to heavy traffic are first calculated using our approximate algorithm. Simulation models and programs are constructed and implemented to obtain the performance measurements for the same system under an identical set of traffic conditions. The results are summarized in table10-12. These tables show that most of our results from our approximate algorithm differ by less than 10% from the simulation results. We observe that the performance measurements differ by larger than 10%(about 14%) in only the case in which the parameters:buffer size= 5,5,5,token = 20 and the memory access time is 0.9.

Many previous researches in the performance of a queuing network were studied by assuming the buffers are infinite only. However, almost all the real-world computer networks require buffers to be finite. A finite-buffer system can cause blocking phenomena We compare the average queue length for both finite and infinite buffer multiprocessor systems by giving different parameters. The results are calculated and summarized in table 13-15. From observing these tables, we find that the discrepancies of the performance between the cases of infinite buffers and finite buffers of relatively small sizes become quite severe when the traffic intensity becomes heavy. Consequently, the infinite-buffer assumption could cause significant errors. For example, consider the parameters for the proposed finite-buffer multiprocessors system as average interarrival time=1, average service times of 0.0475,0.0475,0.95 and buffer sizes of 1,1,8. By applying our approximation method, the average queue lengths in three stages are found to be 0.03254, 0.201178 and 4.059398 respectively. However, when the buffer sizes are assumed to be infinite, the average queue length in the three stages are 0.02369, 0.02369 and 18.0500 respectively.

Large-scale, shared-memory parallel processors use multistage, packet-switched interconnection networks for routing processor-to-memory traffic. This type of architecture should provide high bandwidth and short latency time when memory requests are distributed randomly. But, if even a small percentage of the memory requests are directed to one specific spot, the network becomes congested and its performance quickly degrades. A study by Pfister and Norton[60] shows that heavy accesses to a "hot spot" shared memory module will not only slow down those processors performing the access, but may cause the entire machine to thrash. To remedy the frequent access to a "hot spot" and cause the degradation in the performance of the system, we propose to study if the performance of system can be improved by limiting each processor to accept a maximum number of outstanding memory requests. For comparison purposes, we first calculate the blocking probability without considering this condition for the proposed machine and summarize the results in table 16. We observe that if the traffic become heavy(utilization > 0.9), then the blocking probability increases dramatically for the case of small to moderate buffer sizes. For example, consider the parameters for the proposed finite-buffer multiprocessors system as average interarrival time=1, average service times of .0475, 0.0475,0.9 and buffer sizes of 1,1,8. By applying our approximation method, the blocking probability in the first and the second stages are 0.1344 and 0.38748 respectively. However, when the average service times are 0.0475, 0.0475 and 0.95. The approximate average blocking probabilities in the first and the second stages increase to 0.3789 and 0.6302, respectively. Can the performance of a multistage multiprocessor under heavy traffic be improved by limiting each processor to a maximum number of outstanding memory requests? The performance measurements for this are calculated and summarized in table 1-9. Not surprisingly, the problems of blocking exhibit improvement even for the small and moderate buffer sizes under heavy traffic. Interestingly, as we can also see from these tables, *server utilizations* are not greatly improved by limiting the outstanding memory requests to be a fixed amount. We conjecture that in our proposed multiprocessor system, when the packets are delivered in the tandem fashion, it is subject to higher blocking probabilities for the earlier stage servers if the memory requests are not restricted in comparing to the situation if the requests are limited. However, the servers may have lesser probability of being idle if there are no restricted memory requests. The balancing of these two factors may explain the fact that server utilizations are not improved by restricting the number of memory requests.

In more complicated networks, we still feel that, by restricting the number of outstanding requests to the system components, the *server utilization* and blocking probabilities can be improved

Literatures studies in this area usually assume the transmission time of the switch element and memory access time are exponentially distributed for easier analysis, which does not agree with the real situations. In reality, these service times are constant. In using the analytic model, we can treat the first stage to be M/D/1 model. However, we do not know which existing theoretic models are proper for later stages. As an exercise, we only explore this situation by modifying our program. We summarize and tabulate the performance measurements due to the use of exponentially distributed service time and the use of constant service time for the same structure from light traffic to heavy traffic, in table 17-19.

arrival time=1 buffer size =1, 1, 5 tokens=4

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500		0.00053	0.00053	0.42060	0.00069	0.00069	0.54453	0.02088	0.02088	0.41764	3.11894
0.07000	0.70000	0.00222	0.00222	0.41602	0.00239	0.00239	0.54327	0.03799	0.03799	0.41499	3.08861
0.14000		0.00714	0.00714	0.41197	0.00926	0.00926	0.53669	0.08256	0.08256	0.40271	3.00592
0.04000		0.00071	0.00071	0.50470	0.00091	0.00091	0.67661	0.02425	0.02425	0.44504	3.00034
0.08000	0.80000	0.00384	0.00384	0.50103	0.00492	0.00492	0.67541	0.04007	0.04007	0.44401	2.96714
0.16000		0.00775	0.00775	0.49947	0.00998	0.00998	0.66360	0.08966	0.08966	0.42248	2.88323
0.04500		0.00080	0.00080	0.60925	0.00098	0.00098	0.78492	0.02516	0.02516	0.47325	2.86558
0.09000	0.90000	0.00437	0.00437	0.59782	0.00542	0.00542	0.78463	0.04313	0.04313	0.47127	2.83591
0.18000		0.00882	0.00882	0.58778	0.01059	0.01059	0.78363	0.09868	0.09868	0.45178	2.74544
0.04750		0.00129	0.00129	0.71864	0.00154	0.00154	0.87434	0.02592	0.02592	0.51843	2.70851
0.09500	0.95000	0.00499	0.00499	0.70913	0.00619	0.00619	0.86534	0.04710	0.04710	0.50740	2.67929
0.19000		0.01298	0.01298	0.69633	0.01432	0.01432	0.85327	0.10808	0.10808	0.49544	2.56611

table 1

arrival time=1 buffer size= 1, 1, 5 tokens=7

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500		0.00116	0.00116	1.03757	0.00116	0.00116	1.03768	0.03500	0.03500	0.69992	5.19020
0.07000	0.70000	0.00429	0.00429	1.03750	0.00429	0.00429	1.03783	0.07000	0.06999	0.69977	5.11416
0.14000		0.01483	0.01483	1.03664	0.01483	0.01483	1.03832	0.13999	0.13995	0.69885	4.95487
0.04000		0.00184	0.00184	1.46065	0.00184	0.00184	1.46403	0.03997	0.03997	0.79781	4.65792
0.08000	0.80000	0.00621	0.00621	1.45913	0.00621	0.00621	1.46481	0.07989	0.07989	0.79637	4.57230
0.16000		0.02026	0.02026	1.45164	0.02030	0.02030	1.46552	0.15949	0.15939	0.79117	4.39779
0.04500		0.00370	0.00370	1.71535	0.00389	0.00389	1.82004	0.04074	0.04074	0.80711	4.38806
0.09000	0.90000	0.00824	0.00824	1.64347	0.00879	0.00879	1.75773	0.07902	0.07900	0.78268	4.37935
0.18000		0.01931	0.01931	1.53642	0.02144	0.02144	1.52979	0.15600	0.15600	0.76283	4.35049
0.04750		0.01543	0.01543	2.25485	0.01610	0.01610	2.44612	0.04359	0.04359	0.83865	3.78846
0.09500	0.95000	0.027867	0.02787	2.25035	0.02906	0.02906	2.47248	0.08739	0.08726	0.83262	3.68664
0.19000		0.05950	0.05950	2.19052	0.06192	0.06192	2.45407	0.17521	0.17374	0.81431	3.52722

table 2

arrival time=1 buffer size 1, 1, 5 tokens=10

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500		0.00127	0.00855	1.03758	0.00127	0.00855	1.03769	0.03500	0.03500	0.69993	8.1827
0.07000	0.70000	0.00522	0.01390	1.03680	0.00522	0.01390	1.03711	0.07000	0.07000	0.69978	8.1043
0.14000		0.02279	0.02703	1.03674	0.02279	0.02704	1.03674	0.14000	0.13996	0.69888	7.9346
0.04000		0.00185	0.03235	1.46578	0.00185	0.03235	1.46298	0.04000	0.04000	0.79847	7.6049
0.08000	0.80000	0.00687	0.04057	1.46279	0.00687	0.04057	1.46757	0.08000	0.07999	0.79739	7.5324
0.16000		0.03048	0.05871	1.46029	0.03048	0.05875	1.47219	0.16000	0.15989	0.79353	7.3371
0.04500		0.01176	0.11718	2.21770	0.01176	0.11720	2.28556	0.04500	0.04499	0.87328	6.7036
0.09000	0.90000	0.02233	0.13315	2.21063	0.02233	0.13325	2.29500	0.09000	0.08992	0.86692	6.6590
0.18000		0.05006	0.17343	2.16274	0.05006	0.17434	2.28975	0.18000	0.17906	0.85008	6.5607
0.04750		0.06295	0.32004	3.06656	0.06295	0.32285	3.39145	0.04750	0.04709	0.85899	5.6546
0.09500	0.95000	0.08747	0.37692	2.98124	0.08747	0.38358	3.17714	0.09500	0.09335	0.84876	5.5173
0.19000		0.16727	0.54574	2.66333	0.16727	0.57394	2.66455	0.19000	0.18066	0.82695	5.4261

table 3

arrival time=1 buffer size =1, 1.8 tokens=8

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500		0.00114	0.00114	1.35091	0.00114	0.00114	1.35093	0.03500	0.03500	0.69999	5.87682
0.07000	0.70000	0.00425	0.00425	1.35087	0.00425	0.00425	1.35099	0.07000	0.07000	0.69994	5.80069
0.14000		0.01471	0.01471	1.34960	0.01471	0.01471	1.35048	0.14000	0.14000	0.69939	5.64159
0.04000		0.00151	0.00151	2.03470	0.00167	0.00167	2.03997	0.03999	0.03999	0.79976	5.08253
0.08000	0.80000	0.00687	0.00687	2.03631	0.00687	0.00687	2.03790	0.08000	0.07999	0.79938	4.99058
0.16000		0.03048	0.03048	2.02011	0.03048	0.03048	2.02685	0.15998	0.15998	0.79734	4.80163
0.04500		0.00269	0.00269	2.66252	0.00277	0.00277	2.74961	0.04260	0.04260	0.84790	4.39900
0.09000	0.90000	0.00761	0.00761	2.61416	0.00789	0.00788	2.67306	0.08379	0.08378	0.83302	4.37003
0.18000		0.02106	0.02106	2.55329	0.02233	0.02233	2.60443	0.16002	0.15988	0.81931	4.26538
0.04750		0.00361	0.00361	2.75571	0.00385	0.00385	2.95980	0.04190	0.04190	0.85071	4.30256
0.09500	0.95000	0.00891	0.08913	2.69673	0.00956	0.00956	2.87909	0.08255	0.08253	0.84717	4.27318
0.19000		0.02558	0.02558	2.58875	0.02756	0.02756	2.84176	0.16359	0.16359	0.83302	4.19989

table 4

arrival time=1 buffer size= 1.1.8 tokens=12

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500		0.00127	0.00301	1.35091	0.00127	0.00301	1.35093	0.03500	0.03500	0.69999	9.87482
0.07000	0.70000	0.00522	0.00707	1.35059	0.00522	0.00707	1.35071	0.07000	0.07000	0.69994	9.79718
0.14000		0.02279	0.02908	1.34730	0.02279	0.02909	1.34848	0.14000	0.13996	0.69939	9.62148
0.04000		0.00167	0.01368	2.03957	0.00167	0.01368	2.04014	0.04000	0.04000	0.79978	9.06530
0.08000	0.80000	0.00687	0.02078	2.03667	0.00687	0.02078	2.03817	0.08000	0.08000	0.79941	8.97627
0.16000		0.03048	0.03742	2.02011	0.03048	0.03744	2.02685	0.15998	0.15989	0.79734	8.79478
0.04500		0.00470	0.07167	2.90082	0.00470	0.07168	2.93472	0.04500	0.04500	0.88961	8.04320
0.09000	0.90000	0.01203	0.08338	2.85109	0.01203	0.08340	2.89831	0.90000	0.08998	0.88561	7.98791
0.18000		0.03951	0.11068	2.71934	0.03346	0.11090	2.80005	0.18000	0.17963	0.87406	7.89678
0.04750		0.03254	0.20100	4.05843	0.03254	0.20140	4.32668	0.04750	0.04741	0.89110	6.72202
0.09500	0.95000	0.04961	0.23237	3.95439	0.04962	0.23358	4.26824	0.09500	0.09451	0.88015	6.69397
0.19000		0.09527	0.32035	3.56318	0.09528	0.32708	3.97677	0.18998	0.18609	0.85120	6.79393

table 5

arrival time=1 buffer size 1.1.8 tokens=16

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500		0.00127	0.00301	1.35090	0.00127	0.00301	1.35090	0.03500	0.03500	0.70000	13.8748
0.07000	0.70000	0.00522	0.00710	1.35060	0.00522	0.00710	1.35070	0.07000	0.06999	0.69990	13.7972
0.14000		0.02279	0.02908	1.34730	0.02279	0.02908	1.34850	0.14000	0.14000	0.69940	13.6214
0.04000		0.00167	0.01370	2.03960	0.00167	0.01370	2.04010	0.04000	0.04000	0.79980	13.0652
0.08000	0.80000	0.00687	0.02080	2.03670	0.00687	0.02080	2.03820	0.08000	0.07999	0.79940	12.9762
0.16000		0.03048	0.03744	2.02100	0.03048	0.03750	2.02750	0.16000	0.15989	0.79740	12.7938
0.04500		0.00470	0.07170	2.90080	0.00470	0.07170	2.93470	0.04500	0.04500	0.89990	12.0329
0.09000	0.90000	0.01200	0.08340	2.85200	0.01200	0.08340	2.89830	0.09000	0.08992	0.86691	12.0058
0.18000		0.03951	0.11070	2.71930	0.03951	0.11090	2.80000	0.18000	0.17960	0.87410	11.8968
0.04750		0.03260	0.20120	4.05940	0.03260	0.20160	4.32780	0.04750	0.04740	0.89110	10.7208
0.09500	0.95000	0.04980	0.23300	3.95650	0.04980	0.23420	4.27100	0.09500	0.09450	0.88000	10.6912
0.19000		0.09560	0.32150	3.56300	0.09560	0.32830	3.97740	0.19000	0.18610	0.85100	10.7928

table 6

arrival time=1 buffer size=5,5,5 tokens=10

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500	0.70000	0.00127	0.01166	1.03759	0.00127	0.01166	1.03759	0.03500	0.03500	0.70000	8.17948
0.07000		0.00527	0.02095	1.03759	0.00527	0.02095	1.03759	0.07000	0.07000	0.70000	8.09619
0.14000		0.02279	0.02702	1.03349	0.02279	0.02703	1.03514	0.14000	0.13995	0.69889	7.93786
0.04000	0.80000	0.00167	0.06346	1.46298	0.00167	0.06346	1.46299	0.04000	0.04000	0.79996	7.59193
0.08000		0.00696	0.08772	1.46294	0.00696	0.08772	1.46294	0.08000	0.08000	0.79999	7.48239
0.16000		0.03050	0.15227	1.46279	0.03050	0.15227	1.46279	0.16000	0.16000	0.79994	7.23450
0.04500	0.90000	0.00256	0.41658	2.20169	0.00256	0.41659	2.20525	0.04500	0.04500	0.89855	6.39060
0.09000		0.01026	0.48879	2.19160	0.01026	0.48880	2.19814	0.09000	0.09000	0.89773	6.23206
0.18000		0.03951	0.65392	2.15841	0.03951	0.65397	2.17053	0.17999	0.17999	0.89497	5.89321
0.04750	0.95000	0.01166	0.97173	3.15853	0.01166	0.97703	3.22589	0.04724	0.04724	0.93016	4.83344
0.09500		0.02722	1.00451	3.01341	0.02722	1.01411	3.09496	0.09410	0.09410	0.92497	4.84169
0.19000		0.04475	1.09853	2.79357	0.04475	1.11716	2.90343	0.18683	0.18683	0.91406	4.77543

table 7

arrival time=1 buffer size=5,5,5 tokens=15

service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500	0.70000	0.00127	0.01166	1.03759	0.00127	0.01166	1.03759	0.03500	0.03500	0.70000	13.1795
0.07000		0.00527	0.02095	1.03759	0.00527	0.02095	1.03759	0.07000	0.07000	0.70000	13.0962
0.14000		0.02278	0.04952	1.03758	0.02278	0.04952	1.03758	0.14000	0.14000	0.69999	13.9101
0.04000	0.80000	0.00167	0.06346	1.46298	0.00167	0.06346	1.46299	0.04000	0.04000	0.80000	12.5919
0.08000		0.00696	0.08772	1.46294	0.00696	0.08772	1.46296	0.08000	0.08000	0.79999	12.4824
0.16000		0.03050	0.15223	1.46267	0.03050	0.15227	1.46279	0.16000	0.16000	0.79994	12.2346
0.04500	0.90000	0.00256	0.41663	2.20117	0.00256	0.41663	2.20532	0.04500	0.04500	0.89855	11.3461
0.09000		0.01027	0.48892	2.19178	0.01027	0.48892	2.19729	0.09000	0.09000	0.89774	11.2313
0.18000		0.04603	0.65461	2.15913	0.04603	0.65461	2.17114	0.18000	0.18000	0.89502	10.8852
0.04750	0.95000	0.01677	1.07504	3.38475	0.01677	1.07504	3.45254	0.04750	0.04750	0.93135	10.4970
0.09500		0.04457	1.18922	3.38470	0.04457	1.18922	3.47110	0.09500	0.09500	0.92635	10.2651
0.19000		0.17788	1.45066	3.37917	0.17788	1.45085	3.51345	0.19000	0.18998	0.91369	9.7906

table 8

arrival time=1 buffer size=5,5,5 tokens=20

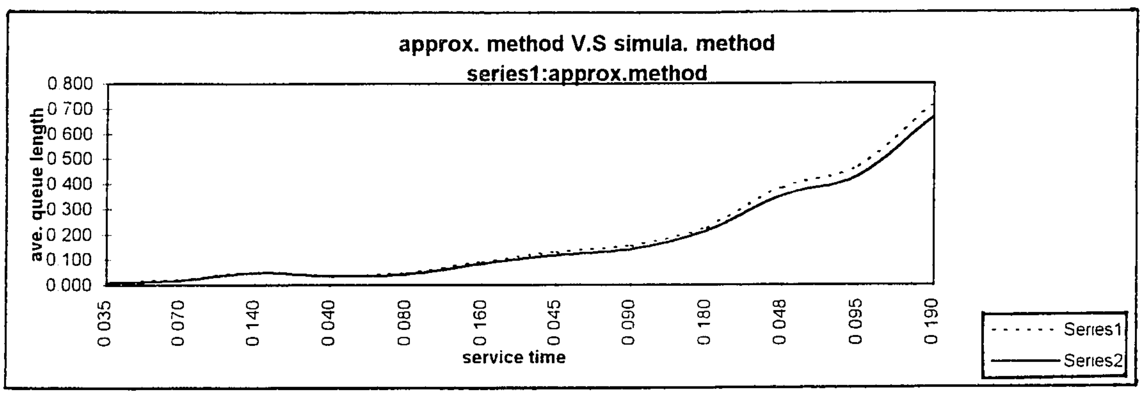
service time		average queue length			average waiting time			server utilization			token in node 0
switch	memory	stage1	stage2	memory	stage1	stage2	memory	stage1	stage2	memory	
0.03500	0.70000	0.00127	0.01166	1.03758	0.00127	0.01166	1.03759	0.03500	0.03500	0.70000	18.1795
0.07000		0.00527	0.02095	1.03759	0.00527	0.02095	1.03759	0.07000	0.07000	0.70000	18.0962
0.14000		0.02278	0.04952	1.03758	0.02278	0.04958	1.03758	0.14000	0.14000	0.69990	17.9102
0.04000	0.80000	0.00167	0.06346	1.46298	0.00167	0.06346	1.46300	0.04000	0.04000	0.80000	17.5919
0.08000		0.00696	0.08772	1.46294	0.00696	0.08722	1.46300	0.08000	0.08000	0.79990	17.4825
0.16000		0.03050	0.15227	1.46266	0.03049	0.15227	1.52270	0.16000	0.16000	0.79990	17.2347
0.04500	0.90000	0.00256	0.41663	2.20177	0.00256	0.41663	2.20532	0.04500	0.04500	0.89855	16.3905
0.09000		0.01027	0.48892	2.19178	0.01027	0.48892	2.19730	0.09000	0.09000	0.89774	16.2313
0.18000		0.04603	0.65461	2.15913	0.04603	0.65606	2.17114	0.18000	0.18000	0.89500	15.8852
0.04750	0.95000	0.01677	1.07504	3.38476	0.01677	1.07504	3.45254	0.04750	0.04750	0.93135	14.4971
0.09500		0.04457	1.18922	3.38470	0.04457	1.18922	3.47111	0.09500	0.09500	0.92635	14.2652
0.19000		0.17789	1.45067	3.37919	0.17789	1.45086	3.51347	0.19000	0.19000	0.91369	13.6986

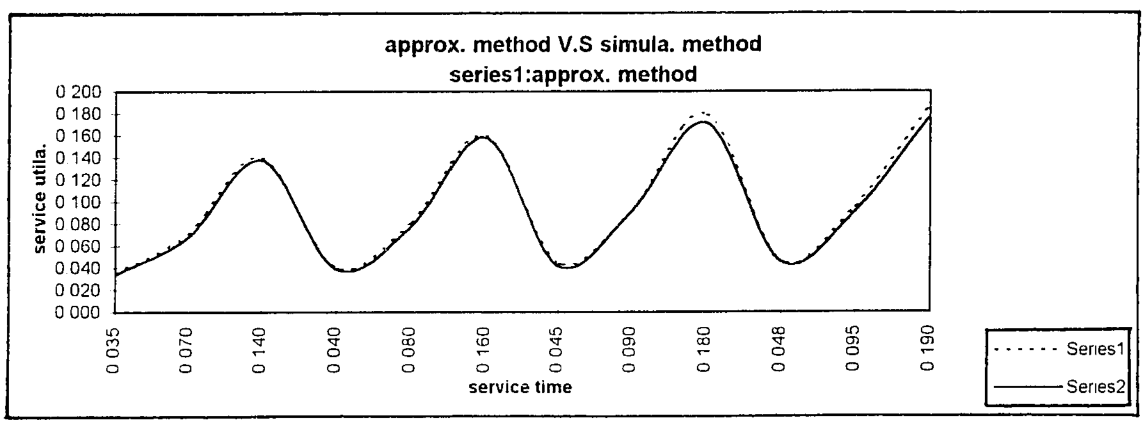
table 9

arrival time=1 buffer size=1 1 5 No. of token=10

service time		average queue length				average waiting time				server utilization			
		approx method		simula. method		approx. method		simula. method		approx. method		simula. method	
switch	memory	switches	memory	switche	memory	switches	memory	switches	memory	switches	memory	switches	memory
0.0350		0.0098	1.0376	0.0105	1.1175	0.0098	1.0377	0.0109	1.1280	0.0350	0.6999	0.0342	0.6999
0.0700	0.7000	0.0191	1.0368	0.0185	1.1135	0.0191	1.0371	0.0195	1.1244	0.0700	0.6998	0.0682	0.6819
0.1400		0.0498	1.0367	0.0502	1.1032	0.0498	1.0367	0.0497	1.1034	0.1400	0.6989	0.1384	0.6893
0.0400		0.0342	1.4658	0.0372	1.5419	0.0342	1.4630	0.0377	1.5523	0.0400	0.7985	0.0388	0.7681
0.0800	0.8000	0.0474	1.4628	0.0450	1.5127	0.0474	1.4676	0.0451	1.5207	0.0800	0.7974	0.0781	0.7658
0.1600		0.0892	1.4603	0.0853	1.4990	0.0892	1.4722	0.0845	1.5615	0.1599	0.7935	0.1580	0.7633
0.0450		0.1289	2.2177	0.1177	2.1310	0.1289	2.2856	0.1261	2.1825	0.0450	0.8733	0.0420	0.8434
0.0900	0.9000	0.1554	2.2106	0.1430	2.0913	0.1556	2.2950	0.1489	2.1225	0.0900	0.8669	0.0915	0.8365
0.1800		0.2235	2.1627	0.2152	2.0514	0.2244	2.2898	0.2577	2.1023	0.1795	0.8501	0.1715	0.8221
0.0475		0.3830	3.0666	0.3530	2.8222	0.3858	3.3915	0.3657	3.1857	0.0473	0.8590	0.0457	0.8211
0.0950	0.9500	0.4644	2.8124	0.4306	2.5998	0.4711	3.1771	0.4317	3.0948	0.0941	0.8444	0.0916	0.8226
0.1900		0.7130	2.2633	0.6678	2.1389	0.7412	2.6646	0.7430	2.6813	0.1854	0.8070	0.1774	0.8001

table 10

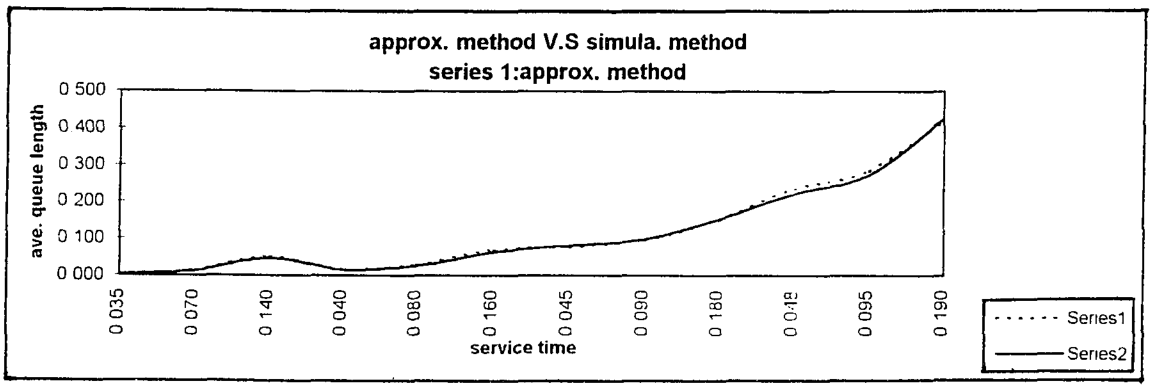


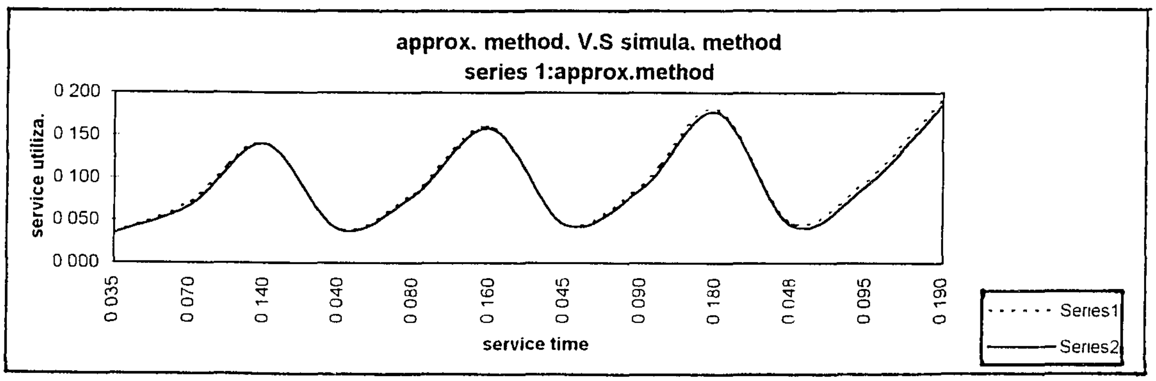


arrival time=1 buffer size = 1,1,8 token=16

service	time	average queue lengt				average waiting time				server utilization			
		approx method		simula. method		approx. method		simula. method		approx. method		simula. method	
switch	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory
0.0350		0.0043	1.3509	0.0043	1.4732	0.0043	1.3509	0.0045	1.4632	0.0350	0.6999	0.0348	0.7030
0.0700	0.7000	0.0123	1.3506	0.0121	1.4244	0.0123	1.3507	0.0119	1.4763	0.0700	0.6999	0.0676	0.6832
0.1400		0.0519	1.3473	0.0483	1.3791	0.0519	1.3485	0.0498	1.4550	0.1400	0.7000	0.1396	0.6987
0.0400		0.0154	2.0396	0.0159	2.1121	0.0154	2.0401	0.0161	2.1043	0.0400	0.7985	0.0398	0.7856
0.0800	0.8000	0.0277	2.0367	0.0271	2.0013	0.0277	2.0382	0.0266	2.0866	0.0800	0.7994	0.0789	0.7659
0.1600		0.0679	2.0210	0.0640	2.0985	0.0680	2.0275	0.0646	2.0421	0.1600	0.7974	0.1579	0.7961
0.0450		0.0764	2.9008	0.0806	3.1760	0.0764	2.9347	0.0811	3.2198	0.0450	0.8896	0.0447	0.8565
0.0900	0.9000	0.0954	2.8511	0.0991	2.9818	0.0954	2.8983	0.1023	3.0952	0.0900	0.8856	0.0874	0.8604
0.1800		0.1502	2.7193	0.1529	2.9224	0.1504	2.8000	0.1574	3.0329	0.1800	0.8741	0.1769	0.8690
0.0475		0.2335	4.0584	0.2222	4.0904	0.2342	4.3278	0.2514	4.4323	0.0475	0.8911	0.0442	0.8809
0.0950	0.9500	0.2820	3.9544	0.2731	4.1192	0.2840	4.2710	0.2750	4.4001	0.0950	0.8800	0.0912	0.8859
0.1900		0.4156	3.5632	0.4260	3.6882	0.4239	3.9774	0.4470	4.1238	0.1900	0.8510	0.1862	0.8682

table 11

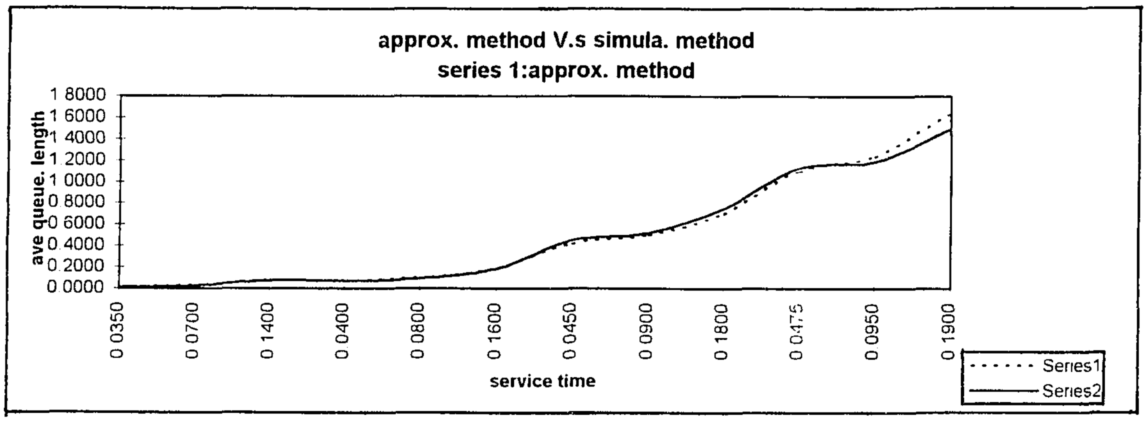


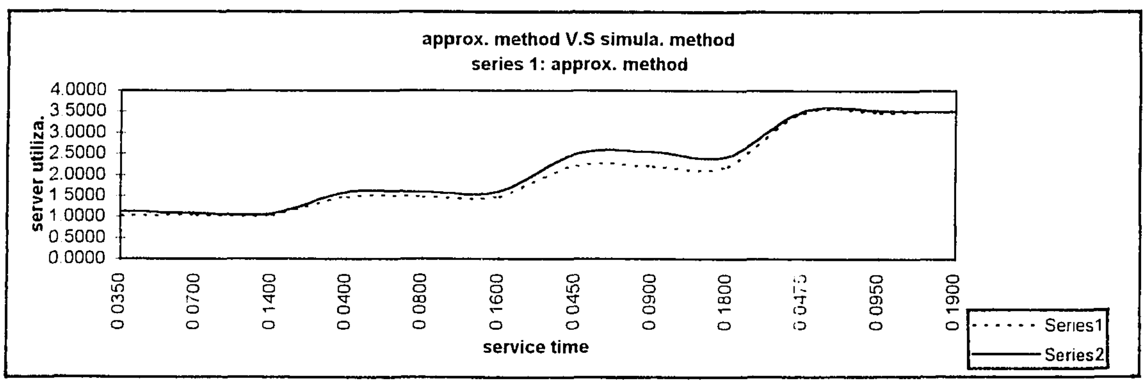


arrival time=1 buffer size=5 5 5 No. of token=20

service time		average queue length				average waiting time				server utilization			
		approx method		simula. method		approx. method		simula. method		approx. method		simula. method	
switch	memory	switches	memory	switche	memory	switches	memory	switches	memory	switches	memory	switches	memory
0.0350	0.7000	0.0130	1.0376	0.0135	1.1506	0.0130	1.0376	0.0140	1.1403	0.0350	0.7000	0.0348	0.7030
0.0700		0.0262	1.0376	0.0271	1.1025	0.0262	1.0376	0.0265	1.1008	0.0700	0.7000	0.0695	0.7039
0.1400		0.0723	1.0376	0.0807	1.0762	0.0723	1.0367	0.0804	1.0771	0.1400	0.7000	0.1399	0.6985
0.0400	0.8000	0.0651	1.4631	0.0708	1.6023	0.0651	1.4631	0.0703	1.5908	0.0400	0.8000	0.0393	0.7924
0.0800		0.0947	1.4629	0.0984	1.5504	0.0947	1.4629	0.1022	1.5880	0.0800	0.8000	0.0786	0.7853
0.1600		0.1828	1.4627	0.1912	1.5701	0.1828	1.4628	0.1919	1.6033	0.1599	0.8000	0.1563	0.7954
0.0450	0.9000	0.4192	2.2018	0.4555	2.5639	0.4192	2.2053	0.4591	2.5087	0.0450	0.8986	0.0441	0.8790
0.0900		0.4992	2.1918	0.5231	2.5367	0.4992	2.1973	0.4949	2.5448	0.0900	0.8977	0.0893	0.8869
0.1800		0.7006	2.1591	0.7542	2.3774	0.7006	2.1711	0.7588	2.4348	0.1795	0.8950	0.1777	0.9019
0.0475	0.9500	1.0918	3.3848	1.1338	3.4732	1.0918	3.4525	1.1532	3.5219	0.0475	0.9313	0.0464	0.9236
0.0950		1.2338	3.3847	1.1916	3.4039	1.2338	3.4711	1.2086	3.5297	0.0950	0.9264	0.0937	0.9180
0.1900		1.6286	3.3792	1.4968	3.2806	1.6286	3.5135	1.5482	3.4985	0.1900	0.9137	0.1840	0.9165

table 12





Average Queuing Length: arrival time = 1

service time		buffer sizes= 1, 1,5			infinite buffer		
swithch	memory	stage1	stage 2	memory	stage1	stage2	memory
0.03500		0.00127	0.00850	1.03758	0.00127	0.00127	1.63333
0.07000	0.70000	0.00522	0.01390	1.03680	0.00527	0.00527	1.63333
0.14000		0.02279	0.02703	1.03674	0.02279	0.02279	1.63333
0.04000		0.00185	0.03235	1.46578	0.00167	0.00167	3.20000
0.08000	0.80000	0.00687	0.04057	1.46279	0.00657	0.00657	3.20000
0.16000		0.03048	0.05871	1.46029	0.03048	0.03048	3.20000
0.04500		0.01176	0.11718	2.21770	0.00212	0.00212	8.10000
0.09000	0.90000	0.02233	0.13315	2.21063	0.00890	0.00890	8.10000
0.18000		0.05006	0.17343	2.16274	0.03951	0.03951	8.10000
0.04750		0.06295	0.32004	3.06656	0.00237	0.00237	18.05000
0.09500	0.95000	0.08747	0.37692	2.81238	0.00997	0.00997	18.05000
0.19000		0.16727	0.54574	2.26333	0.04457	0.04457	18.05000

table 13

Average Queuing Length: arrival time = 1

service time		buffer sizes= 1,1,8			infinite buffer		
swithch	memory	stage1	stage 2	memory	stage1	stage2	memory
0.03500		0.00127	0.00301	1.35090	0.00127	0.00127	1.63333
0.07000	0.70000	0.00522	0.00710	1.35060	0.00527	0.00527	1.63333
0.14000		0.02279	0.02908	1.34730	0.02279	0.02279	1.63333
0.04000		0.00167	0.01370	2.03960	0.00167	0.00167	3.20000
0.08000	0.80000	0.00687	0.02080	2.03670	0.00657	0.00657	3.20000
0.16000		0.03048	0.03744	2.02100	0.03048	0.03048	3.20000
0.04500		0.00470	0.07170	2.90080	0.00212	0.00212	8.10000
0.09000	0.90000	0.01200	0.08340	2.85200	0.00890	0.00890	8.10000
0.18000		0.03951	0.11070	2.71930	0.03951	0.03951	8.10000
0.04750		0.03260	0.20120	4.05940	0.00237	0.00237	18.05000
0.09500	0.95000	0.04980	0.23300	3.95650	0.00997	0.00997	18.05000
0.19000		0.09560	0.32150	3.56300	0.04457	0.04457	18.05000

table 14

Average Queuing Length: arrival time = 1

service time		buffer sizes= 5, 5,5			infinite buffer		
swithch	memory	stage1	stage 2	memory	stage1	stage2	memory
0.03500	0.70000	0.00127	0.01166	1.03758	0.00127	0.00127	1.63333
0.07000		0.00527	0.02095	1.03759	0.00527	0.00527	1.63333
0.14000		0.02278	0.04952	1.03758	0.02279	0.02279	1.63333
0.04000	0.80000	0.00167	0.06346	1.46298	0.00167	0.00167	3.20000
0.08000		0.00696	0.08772	1.46294	0.00657	0.00657	3.20000
0.16000		0.03050	0.15227	1.46266	0.03048	0.03048	3.20000
0.04500	0.90000	0.00256	0.41663	2.20177	0.00212	0.00212	8.10000
0.09000		0.01027	0.48892	2.19178	0.00890	0.00890	8.10000
0.18000		0.04603	0.65461	2.15913	0.03951	0.03951	8.10000
0.04750	0.95000	0.01677	1.07504	3.38476	0.00237	0.00237	18.05000
0.09500		0.04457	1.18922	3.38470	0.00997	0.00997	18.05000
0.19000		0.17789	1.45067	3.37919	0.04457	0.04457	18.05000

table 15

Blocking Probability

arrival time	service time		buffer = 1,1,5		buffer = 1,1,8		buffer = 5,5,5	
	switch	memory	stage1	stage2	stage1	stage2	stage1	stage2
1.00	0.035		0.0105	0.1176	0.0034	0.0404	0.0001	0.1176
1.00	0.070	0.7	0.0181	0.1176	0.0085	0.0404	0.0001	0.1176
1.00	0.140		0.0397	0.1176	0.0257	0.0404	0.0001	0.1176
1.00	0.040		0.0495	0.2621	0.0178	0.1342	0.0001	0.2621
1.00	0.080	0.8	0.0656	0.2621	0.0290	0.1342	0.0003	0.2621
1.00	0.160		0.1047	0.2621	0.0594	0.1342	0.0011	0.2621
1.00	0.045		0.2364	0.5314	0.1344	0.3874	0.0132	0.5314
1.00	0.090	0.9	0.2684	0.5314	0.1617	0.3874	0.0193	0.5314
1.00	0.180		0.3384	0.5314	0.2236	0.3874	0.0388	0.5314
1.00	0.048		0.4942	0.7283	0.3789	0.6302	0.1282	0.7351
1.00	0.095	0.95	0.5368	0.7288	0.4190	0.6302	0.1632	0.7351
1.00	0.190		0.6266	0.7292	0.5054	0.6302	0.2574	0.7351

table 16

arrival time=1 buffer size=1 1 5 No. of token=10

service time		average queue length				average waiting time				server utilization			
		deter. service		expon service		deter. service		expon. service		deter. service		expon. service	
switch	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory
0.0350		0.0044	1.2328	0.0105	1.1175	0.0043	1.2543	0.0109	1.1280	0.0340	0.6884	0.0342	0.6999
0.0700	0.7000	0.0052	1.2090	0.0185	1.1135	0.0056	1.2112	0.0195	1.1244	0.0684	0.6826	0.0682	0.6819
0.1400		0.0150	1.0514	0.0502	1.1032	0.0152	1.0369	0.0497	1.1034	0.1401	0.7024	0.1384	0.6893
0.0400		0.0073	1.6806	0.0372	1.5419	0.0076	1.6708	0.0377	1.5523	0.0373	0.7579	0.0389	0.7681
0.0800	0.8000	0.0121	1.5820	0.0450	1.5127	0.0122	1.6661	0.0451	1.5207	0.0772	0.7729	0.0781	0.7658
0.1600		0.0196	1.4937	0.0853	1.4990	0.0204	1.5661	0.0845	1.5615	0.1509	0.7570	0.1580	0.7633
0.0450		0.0200	2.1483	0.1177	2.1310	0.0218	2.2043	0.1261	2.0825	0.0413	0.8480	0.0420	0.8434
0.0900	0.9000	0.0218	1.9845	0.1430	2.0913	0.0266	2.1400	0.1489	2.0725	0.0857	0.8401	0.0915	0.8365
0.1800		0.0409	1.9251	0.2152	2.0514	0.0455	2.0493	0.2577	1.8104	0.1728	0.8289	0.1715	0.8221
0.0475		0.0236	2.8342	0.3530	2.8222	0.0242	3.2596	0.3657	3.1857	0.0435	0.8232	0.0457	0.8211
0.0950	0.9500	0.0282	2.5870	0.4306	2.5998	0.0340	3.1380	0.4317	3.0948	0.0874	0.8268	0.0916	0.8226
0.1900		0.0417	2.3387	0.6678	2.1389	0.0424	2.7685	0.7430	2.6813	0.1769	0.7991	0.1774	0.8001

table 17

arrival time=1 buffer size = 1.1.8 token=16

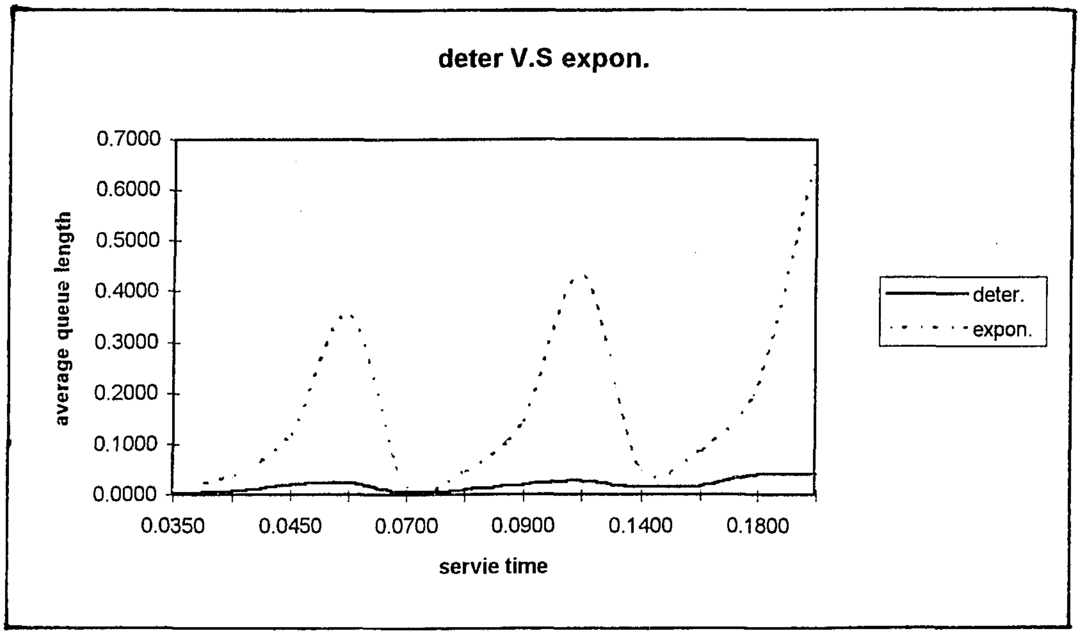
service	time	average queue lengt				average waiting time				server utilization			
		deter. serivce		expon. service		deter. service		expon. service		deter. service		expon. service	
switch	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory
0.0350		0.0043	1.6061	0.0043	1.4732	0.0047	1.5865	0.0045	1.4632	0.0351	0.7027	0.0348	0.7030
0.0700	0.7000	0.0064	1.5831	0.0121	1.4244	0.0061	1.5703	0.0119	1.4763	0.0700	0.6999	0.0693	0.6832
0.1400		0.0127	1.3464	0.0483	1.3791	0.0131	1.3547	0.0498	1.4550	0.1417	0.7078	0.1396	0.6987
0.0400		0.0076	2.2569	0.0159	2.1121	0.0076	2.1758	0.0161	2.1043	0.0397	0.7967	0.0398	0.7856
0.0800	0.8000	0.0136	2.1829	0.0271	2.0013	0.0132	2.0195	0.0266	2.0866	0.0797	0.7950	0.0789	0.7659
0.1600		0.0342	2.1061	0.0640	2.0985	0.0332	2.1259	0.0646	2.0421	0.1595	0.7976	0.1579	0.7961
0.0450		0.0376	3.8073	0.0806	3.1760	0.0373	3.8467	0.0811	3.2198	0.0452	0.8996	0.0447	0.8565
0.0900	0.9000	0.0440	3.2830	0.0991	2.9818	0.0445	3.3661	0.1023	3.0952	0.0896	0.8930	0.0874	0.8604
0.1800		0.0479	3.1217	0.1529	2.9224	0.0498	3.1408	0.1574	3.0329	0.1743	0.8834	0.1769	0.8690
0.0475		0.0475	4.1946	0.2222	4.0904	0.0521	4.3278	0.2514	4.4323	0.0453	0.8964	0.0442	0.8809
0.0950	0.9500	0.0524	4.2759	0.2731	4.1192	0.0543	4.6119	0.2750	4.4001	0.0910	0.9063	0.0912	0.8859
0.1900		0.0609	3.8434	0.4260	3.6882	0.0615	4.0655	0.4470	4.1238	0.1888	0.9199	0.1862	0.8682

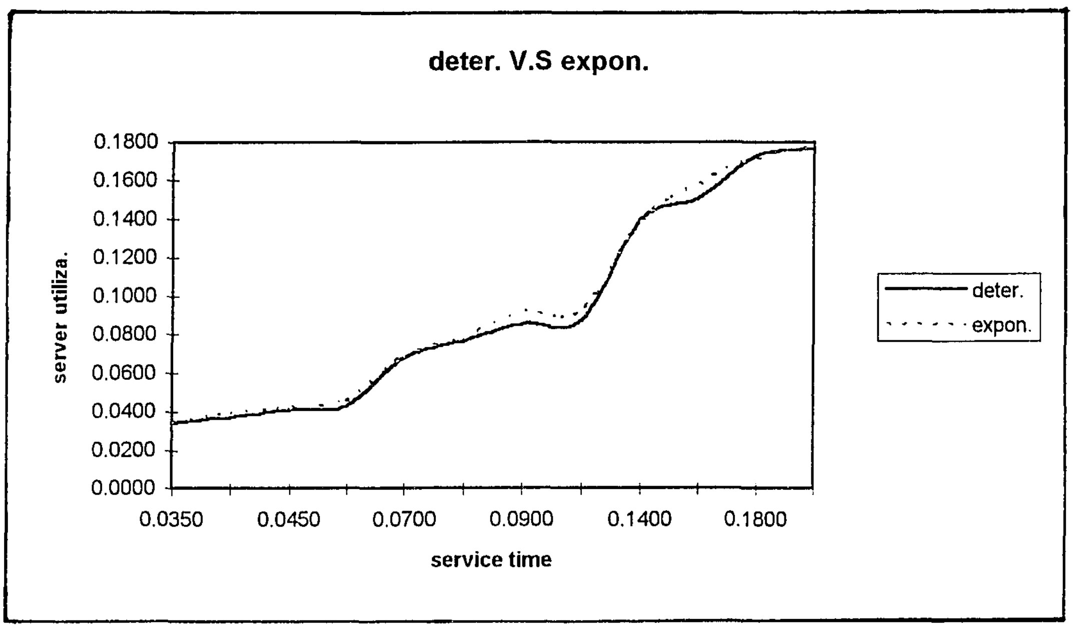
table 18

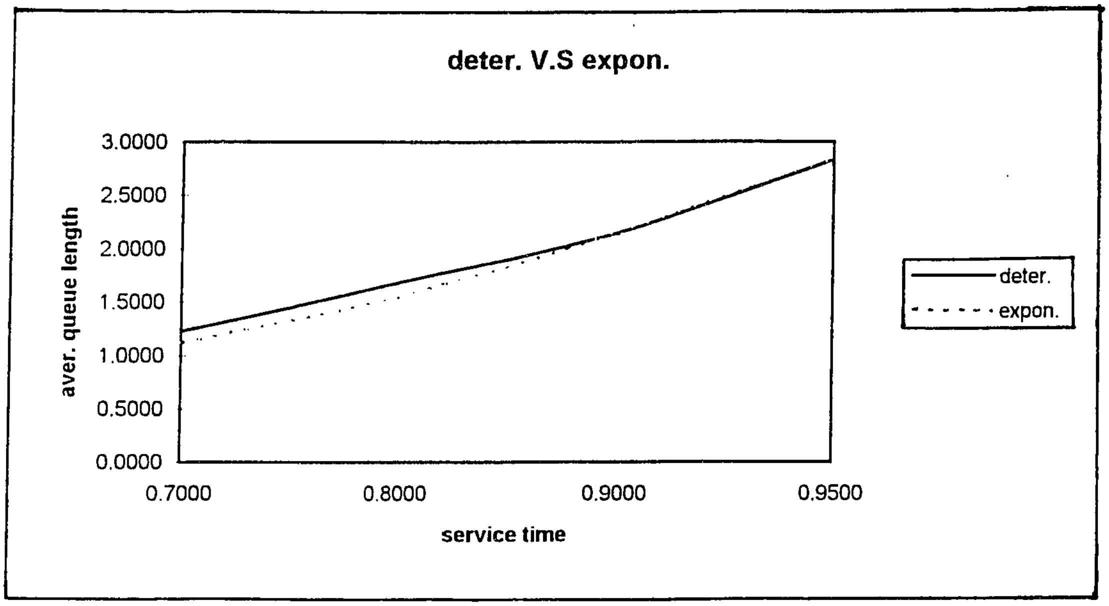
arrival time=1 buffer size=5 5 5 No. of token=20

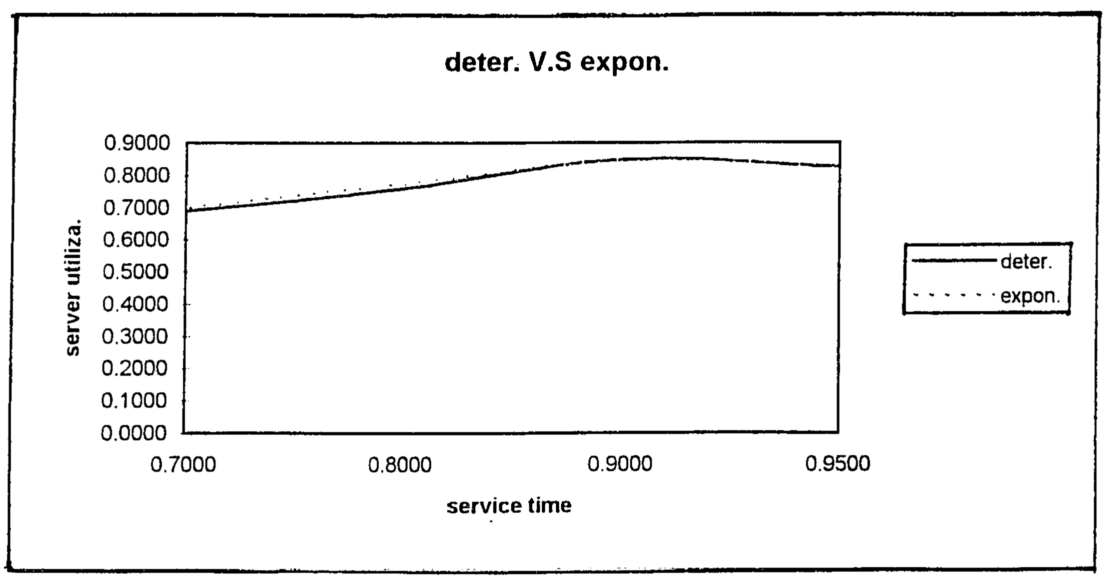
service time		average queue length				average waiting time				server utilization			
		deter. service		expon service		deter. service		expon. service		deter. service		expon. service	
switch	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory	switches	memory
0.0350	0.7000	0.0053	1.2506	0.0135	1.1506	0.0053	1.2293	0.0140	1.1403	0.0355	0.7110	0.0348	0.7030
0.0700		0.0096	1.1888	0.0271	1.1025	0.0102	1.2100	0.0265	1.1008	0.0715	0.7092	0.0695	0.7039
0.1400		0.0225	1.0149	0.0807	1.0762	0.0244	1.0213	0.0804	1.0771	0.1414	0.7053	0.1399	0.6985
0.0400	0.8000	0.0119	1.6714	0.0708	1.6023	0.0121	1.7246	0.0703	1.5908	0.0387	0.7773	0.0393	0.7924
0.0800		0.0149	1.6161	0.0984	1.5504	0.0144	1.6472	0.1022	1.5880	0.0802	0.8041	0.0786	0.7853
0.1600		0.0321	1.5431	0.1912	1.5701	0.0311	1.5632	0.1919	1.6033	0.1608	0.8051	0.1563	0.7954
0.0450	0.9000	0.0762	2.6317	0.4555	2.5639	0.0792	2.6404	0.4591	2.5087	0.0449	0.9007	0.0441	0.8790
0.0900		0.0980	2.5596	0.5231	2.5367	0.1012	2.5839	0.4949	2.5448	0.0885	0.8842	0.0893	0.8869
0.1800		0.1238	2.4222	0.7542	2.3774	0.1233	2.4096	0.7588	2.4348	0.1798	0.8986	0.1777	0.9019
0.0475	0.9500	0.2137	3.3692	1.1338	3.4732	0.2251	3.3803	1.1532	3.5219	0.0473	0.9390	0.0464	0.9236
0.0950		0.3392	3.2267	1.1916	3.4039	0.3342	3.2946	1.2086	3.5297	0.0936	0.9326	0.0937	0.9180
0.1900		0.6550	3.2454	1.4968	3.2806	0.6663	3.2492	1.5482	3.4985	0.1905	0.9422	0.1840	0.9165

table 19


deter V.S expon.
average queue length
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
0.0350
0.0450
0.0700
0.0900
0.1400
0.1800
servie time
deter.
expon.


deter. V.S expon.
server utiliza.
0.1800
0.1600
0.1400
0.1200
0.1000
0.0800
0.0600
0.0400
0.0200
0.0000
0.0350
0.0450
0.0700
0.0900
0.1400
0.1800
service time
deter.
expon.


deter. V.S expon.
aver. queue length
3.0000
2.5000
2.0000
1.5000
1.0000
0.5000
0.0000
0.7000
0.8000
0.9000
0.9500
service time
deter.
expon.


deter. V.S expon.
server utiliza.
0.9000
0.8000
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
0.0000
0.7000
0.8000
0.9000
0.9500
service time
deter.
expon.

Chapter 8
Conclusions and Future Work

In this dissertation, we have explored three major areas: We have developed and presented an approximation algorithm to estimate the performance of an multistage, asynchronous, packet-switched, finite buffer multiprocessor system with limited outstanding memory requests. Scale relationships for the various performance measurements affected by:the number of processors, a characterization of cache memory limitation on number of memory requests, the number of memory modules, the memory access time and buffer size, the topology and characteristics of the interconnection network are developed and studied. Furthermore, the algorithm we present in this dissertation, can be extended to estimate the performance of any finite buffer banyan network.

In chapter 7, we verified the accuracy of our approximation algorithm using a simulation study. In our numerical examples, the results of the approximate algorithm and those of simulation method mostly differ less than 9%. Only one case reaches 14%.

Our second area of study is to compare performances of a finite and infinite buffer multiprocessor machine and to see how the performances are effected by the buffer size, the packet arrival rate and the packet transmitted rate. It was found that for the value used a finite-buffer machine has nearly the same performance as an infinite buffer machine does when the traffics are light to moderate and the buffer sizes are moderate. However, once the traffic load becomes heavy (arrival rate/service rate > 0.9) and buffer size is relative small(buffer size $\leq$ 5), the blocking probability increases rapidly. This phenomena was observed in[1][2], but the calculation of blocking probability was not given. We have derived an approximation formulas for them. The approximation formula can be applied to obtain the blocking probability in each stage. The results we obtain from the approximation formulas are verified by simulation. Our study shows that the approximation formulas can provide an efficient and good approximation to calculate the blocking probability. In our experimental results with a, utilization of 0.95 and buffer size of 5, the blocking probability can reach as high as 73.5%.

Thirdly, we did a study to see if the performance of the proposed multiprocessor machine can be improved by restricting outstanding memory requests. We model the proposed multiprocessor by a closed queuing network and subsequently, derive approximation formulas for it. Performance measurements calculated by our approximation formulas are verified by using simulation. In the tested examples, we observed that the blocking probability of the proposed machine is reduced compared to the machine without restricting memory requests. However, the server utilizations of the proposed machine are not improved by restricting the outstanding memory requests.

Furthermore, a lot of effort and time was spent in the design and coding of the simulation program. We feel that this program can be easily extended to provide for the performance analysis of any blocking type(service-before-blocking, service-after-blocking), any buffer size(finite or infinite) and any service time distribution(e.g. exponentially or constant service time, ..) of a queuing network. We hope the program can become a useful tool in our future study in this area.

The program will be sent to any interested reader.

In the future, we intend to continue this research. A network with feedback is expected to complicate the problem considerably, and yet it has other applications, such as transportation network, and communication network.s.

Bibliography

[1] Akyildiz I. F. "Exact product form solution for queueing network with blocking". *IEEE Trans. on Computer*, pp. 122-125, Jan. 1987.

[2] Akyildiz I. F. "On the exact and approximate throughput analysis of closed queueing networks with blocking". *IEEE Trans. Softw. Eng.*, SE-14, pp. 62-71, 1988.

[3] Akyildiz I. F. "Mean value analysis for blocking queueing networks". *IEEE Trans. Softw. Eng.*, SE-14 (1), pp. 418-429, 1988.

[4] Almasi G. S. and Gottlieb A., Highly Parallel Computing. *Benjamin/Cummings Publishing Inc.* 1989.

[5] Allen A. O., Probability, Statistics, and Queueing Theory with Computer Science Application. *Computer Science and Scientific Computing Inc.* Second Edition.

[6] Bailey D. H. "Vector computer memory bank contention". *IEEE Trans. Comput.*, C-36 pp. 293-298, 1987.

[7] Balsamo S. Clo M. C. and Donatiello "Cycle time distribution of cyclic networks with blocking". *Performance Evaluation*, pp. 159 168 may 1993.

[8] Baskett F.,Chandy K.M.,Muntz R.R., and Palacios F.G."Open, closed and mixed networks of queues with different classes of customers". J. *ACM* 22, pp. 248-260, 1975.

[9] Bucher I. Y. and Calahan D. A. "Access conflicts in multiprocessor memories queueing models and simulation studies". *Proceeding 1990 ACM Sigmetrics*, pp. 428-438, Aug. 1990.

[10] Buzen J. P and Denning P. J. "Operational treatment of queue distributions and mean-value analysis". *Computer Performance*, pp. 6-15, June 1980.

[11] Calahan D. A. and Bailey D. A. "Measurement and analysis of memory conflicts on a vector multiprocessor". *Performance Evaluation Of Supercomputers*, J. L. Martin, Ed., Elsevier, 1988.

[12] Chang D. Y., Kuck D. J., Lawrie D.H, "On the effective bandwidth of parallel memories". *IEEE Trans. Comput.*, pp. 480-490, May 1977.

[13] Chen W. T. and Shen J. P., "Performance analysis of multistage interconnection networks with hierarchical requesting model". *IEEE Trans. Comput.*, pp. 1438-1442, Nov. 1988.

[14] Clos C. "A study of non-blocking switching networks". *Bell Sys. Tech. Jour.*, pp. 443-468, May 1952.

[15] Cvetanovic Z. "Best and worst mappings for the omega network". *IBM J. RES. Develop.*, pp. 452-463, July 1987.

[16] Dallery Y. "Approximate analysis of general open queueing networks with restricted capacity". *Performance Evaluation* pp. 209-222, 11 1990.

[17] Dan A. and Yu P. S. "Performance analysis of buffer coherency policies in a multisystem data sharing environment". *IEEE Trans. on Parallel and Distributed System.*, pp.289-305. March 1993.

[18] Dandamudi S. P. and Eager D. L. "The influence of locality in routing on queueing networks". *Performance Evaluation*, pp. 93-106, Nov. 1990.

[19] Daniel A. M. and Luiz A. B. "A methodology for performance evaluation of parallel applications on multiprocessors". *Journal of Parallel and Distributed Computing* 14, pp.1-14, 1992.

[20] Decreusefond L., Korezlioglu H. and Van Dijk N.M."An error bound for infinite approximations of queueing networks with large finite stations under RSRD protoco". *Performance Evaluation*, pp. 177-187 May 1993.

[21] Dias D. M.and Jump J. R. "Analysis and simulation of buffered delta networks". *IEEE. Trans. Comput.*, PP. 273-282, April 1981.

[22] Frein Y. and Dallery Y. "Analysis of cyclic queueing networks with finite buffers and blocking before service". *Performance Evaluation* 10, pp. 197-210, 1989.

[23] Garofalakis J. D. and Spirakis P. G. "The performance of multistage interconnection networks with finite buffers". *Proceedings 1990 ACM Sigmetrics.*, pp. 263-264, May 1990.

[24] Gelenbe E. and Mitrani I. Analysis and Synthesis of Computer Systems. *Academic Press* 1980.

[25] Ginosar Ran and Egozi David "Topological comparison of perfect shuffle and hypercube". *International Journal of Parallel Programming*, pp. 37-67, July 1989.

[26] Goke L.R. and Lipovski G. J. "Banyan networks for partitioning multiprocessor system". *The Proceedings of the First Annual Symposium Computer Architecture.*, pp. 21-28, 1973.

[27] Gordon W.J. and Newell G.F. "Closed queueing system exponential servers". *Operation Research*, 15 (2), pp. 254-265 1967.

[28] Gross D. and Harris C. M. Fundamentals of Queueing Theory in Probability and Mathematical Statistics. Wesley series.

[29] Hariri S. and Mohamed A. G. "Modeling availability of parallel computers". *International Conference on Parallel Processing.*, pp. 559-560, 1991.

[30] Harrison P. G. and Patel N. M. "The representation of multistage interconnection networks in queueing models of parallel systems". *ACM,* pp. 863-898, Oct. 1990.

[31] Harrison P. G. and Patel N. M. Performance Modelling of Communication Networks and Computer Architectures. *Addison-Wesley.*

[32] Harrison P. G. and Pinto A. de C. "The approximate analysis of asynchronous, packet_switched buffered banyan network with blocking". *Performance Evaluation* 19, pp. 223-258, April 1994.

[33] Hsieh C.T. and Lam S.S. "PAM-A noniterative approximate solution method for closed multichain queueing networks". *Performance Evaluation* 9, pp. 119-133.

[34] Huang J.H. and Kleinrock L. "Performance evaluation of dynamic sharing of processors in two-stage parallel processing systems". *IEEE. Trans. on Parallel and Distributed Systems.*, pp. 306-317 march 1993.

[35] Hwang K. and Briggs F. A. Computer Architecture and Parallel Processing. *Mcgraw-Hill.*

[36] Ibe O. C. , Chio H. and Trivedi K.S. "Performance evaluation of client-server system". *IEEE Trans. on Parallel and Distributed systems.*, pp. 1217-1229, Nov. 1993.

[37] Jackson J. R. "Networks of waiting line". *Operations Research 5* pp. 518-521, 1957.

[38] Jackson J. R. "Jobshop-like queueing systems". *Management Science 10* pp. 131-142, 1963.

[39] Jaques Labetoulle and Guy Pujolle "Isolation method in a network of queues". *IEEE on Software Eng.*, vol. Se-6, No.4 pp. 373-381, July 1980.

[40] Jianxun D. and Laxmi N. B. "Performance evaluation of multistage interconnection networks with finite buffers". *International Conference on Parallel Processing.* pp I-592 1991.

[41] Kobayashi H. and Konheim A. G. "Queueing models for computer communications system analysis". *IEEE Trans. Communication*, pp. 2-27, Jan. 1977.

[42] Kruskal C. P., Rudolph L. and Snir M. "Efficient synchronization on multiprocessor with shared memory". *ACM Trans. on Programming Lang. and Systems.*, pp. 579-601, Oct. 1988.

[43] Kruskal C. P. and Snir M. "The performance of multistage interconnection networks for multiprocessors". *IEEE Trans. Comput.*, pp. 1091-1098, Dec. 1983.

[44] Kruskal C. P., Snir M. and Weiss A. "The distribution of waiting times in clocked multistage interconnection networks". *IEEE Trans. Comput.*, Vol. C-37, pp. 1145-1155, Nov. 1988.

[45] Kuehn J. P., "Approximate analysis of general queueing networks by decomposition". *IEEE Trans. Communication*, Vol. C-27, pp. 113-126, Jan. 1979.

[46] Lam S.S. "Queueing networks with population size constraints". *IBM J. Res. Develop.*, pp. 370-377, March 1977.

[47] Lavenberg S. S. "A perspective on queueing models of computer performance". *Performance Evaluation*, pp. 53-76, Oct. 1989.

[48] Lawrie D. H. "Access and alignment of data in an array processor". *IEEE Trans. Comput.*, pp. 1145-1155, Dec. 1975.

[49] Lazowska E. D., Zahorjan J., Graham G. S., Sevcik, K.C Quantitative System Performance. *Prentice Hall, New Jersey, 1984.*

[50] Marie R. A. "An approximate analytical method for general queueing networks". *IEEE Trans. Software Eng.*, pp. 530-538, Sept. 1979.

[51] Mitrani I. Modelling of Computer and Communication Systems. *Cambridge University Press* 1987.

[52] Nelson R. D."The mathematics of product form queuing networks". ACM Computing Surveys. pp 339-366., Sept. 1993

[53] Onvural R. O. and Perros H.G., "On equivalences of blocking mechanisms in queueing networks with blocking", operation research lett. 5, (6), pp. 293-297, 1986.

[54] Onvural R. O. and Perros H. G., "Some Equivalencies between closed queueing network with blocking", *Performance Evaluation* 9 pp. 111-118, 1989.

[55] Onvural R. O. and Perros H. G., " Equivalencies between open and closed queueing networks with finite buffers". *Performance Evaluation* 9, pp 263-269, 1989.

[56] Onvural R. O. "Survey of closed queueing networks with blocking". *ACM Computing Surveys*, pp. 85-121, June 1990.

[57] Pawlikowski K. "Steady-state simulation of queueing processes:A survey of problems and solutions". ACM *Computing Surveys*, pp. 123-170,june 1990.

[58] Patel J. H. "Performance of processor-memory interconnections for multiprocessors". *IEEE Trans. Comput.*, pp. 771-780, Oct. 1981.

[59] Pease M. C. III, "The indirect binary n-cube microprocess array". *IEEE Trans. Comput.*, pp. 458-473, May 1977.

[60] Pfister G. F. and Norton V. A. "'Hot spot' contention and combining in multistage interconnection networks". *Proceedings of the 1985 International Conference on Parallel Processing*, pp. 790-797, Aug. 1985.

[61] Reiser M. and Lavenberg S.S, "Mean value analysis of closed queueing networks". J. ACM 27 pp. 313-323, Feb. 1980.

[62] Stone H. S. High-Performance Computer Architecture Addison-Wesley 1987.

[63] Stone H. S. "Parallel processing with the perfect shuffle". *IEEE Trans. Comput.*, pp. 81-89, Feb. 1971.

[64] Suri R. and Diehl G. W., "A variable buffer-szie model and its use in analyzing closed queueing networks with blocking", *Management Science*, Vol. 32, No. 2, pp. 206-225, Feb. 1986.

[65] Szymanski T. H. and Hamcher V. C. "On the permutation capability of multistage interconnection networks". *IEEE Trans. Comput.*, pp. 810-821, July 1987.

[66] Whitt W. "The queuing network analyzer". Bell System Tech. J. pp.2779-2813, 62 (9) 1983.

[67] Willick D. L. and Eager D. L. "An analytical model of multistage interconnection networks". *Proceedings 1990 ACM Sigmetrics*, pp. 192-199, May 1990.

[68] Wu C. L. and Feng T. Y. "On a class of multistage interconnection network". *IEEE on Computer*, pp. 694-702, Aug. 1980.

[69] Yalamanchili S. and Aggarwal J. K. "A characterization and analysis of parallel processor interconnection networks". *IEEE Trans. Comput.*, pp. 680-691, June 1987.

[70] Yu P. S. and Dias D. M. " Performance analysis of concurrency control using locking with defered blocking". *IEEE Trans. on Software Engineering.*, pp 982-996 Oct. 1993.