1997

Performance and Analysis of Segmented Multiple Bus Systems.

Jungjoon Kim
Louisiana State University and Agricultural & Mechanical College

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

PERFORMANCE AND ANALYSIS OF SEGMENTED MULTIPLE BUS SYSTEMS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Jungjoon Kim
B.S., Kyungpook National University, 1981
M.S., Korea Advanced Institute of Science and Technology, 1983
August 1997

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my major advisor, Dr. Ahmed El-Amawy, who has contributed significantly to my professional development by shaping my academic goals, motivating me and supporting my efforts. I am thankful for his insights, suggestions and patient supervision throughout this work.

My sincere appreciation is extended to my committee members, Dr. Alexander Skavantzos, Dr. Ramachandran Vaidyanathan, Dr. Gilsik Lee, Dr. Si-Qing Zheng and Dr. Roger Seals for their encouragement, advice and valuable comments throughout this research. I also thank all Electrical Engineering Department staff for their assistance and support.

I would like to remember all pleasant memories at LSU. I am grateful to wonderful friends who made my life here enjoyable and endurable. I am also grateful for the financial support from Korea Telecom. Thanks to them all.

I am thankful to my parents for their love and sacrifices for me, my brothers and sister. I must add thanks to my parents in law for their prayers and support. To all my beloved ones in Korea who have prayed for me, I am in debt. This dissertation is theirs too.

Finally, I would like to thank my wife, Yoonkyung, for her never-ending support, patience, encouragement and love during this long journey. My beloved children, Seokwon and Seokhyun, deserve sharing this accomplishment. They have always been a great source of encouragement and love.

TABLE OF CONTENTS

List of Tables

List of Figures

ABSTRACT

The dissertation introduces a new class of bus-based systems called the Segmented Multiple Bus System (SMBS). One of the unique characteristics of the SMBS is that it allows the exploitation of memory reference locality even though it is a bus-based nondirect network. This comes from the architectural interconnection feature that the SMBS can be viewed as a large-scale multiple bus system (MBS) that has been partitioned into smaller partitions called segments. Each such segment is in effect a small conventional MBS whose size is chosen so as to avoid bus loading problems. The SMBS overcomes the architectural limitations of bus-based shared memory systems while maintaining their advantages in terms of high degree of fault tolerance, ease of expansion and ease of programming. In addition SMBS's are scalable; unlike conventional MBS's. Another interesting feature of the SMBS is that it supports wormhole routing which is traditionally used in direct network topologies.

We develop performance models to study the SMBS with wormhole routing. Ours is the first attempt to adapt wormhole routing to a bus-based nondirect network. In our performance modeling, features of both direct and nondirect networks are incorporated. We include the effect of blocking and pipelining properties of wormhole routing in the analysis. The bus group of each segment is modeled as a flow equivalent service center which represents a load dependent service center. Two performance models, one assuming single flit buffers and the other assuming infinite flit buffers at segment switches, are developed. Using approximate Mean Value Analysis, we evaluate performance in terms of processing efficiency and request re-

sponse time. We also simulate the two models without applying any approximations. We report comparisons of analytical results with simulation results to support the accuracy and appropriateness of our new and novel performance models. The results demonstrate good match between simulation and analytical results and show good scalability for the SMBS.

The approach we adopt in developing the models is comprehensive in the sense that the models incorporate features of both direct and nondirect networks. This makes our models easily adaptable to several other network topologies.

CHAPTER 1

INTRODUCTION

Massively parallel computers are considered the most promising technology to achieve significant computational power. A great deal of attention has been paid to the design of multiprocessor systems to achieve higher computational power. In such large-scale systems, each processor typically comprises cache memory, local memory and other supporting devices. The processors can execute the same instruction, broadcasted from a control unit, but operate on different data sets from distinct data streams (single instruction stream multiple data stream, or SIMD in short) or they can execute several independent programs simultaneously (multiple instruction stream multiple data stream, or MIMD in short).

The performance of a multiprocessor system depends significantly on its interconnection network. Wide variety of interconnection networks have been proposed for multiprocessor systems [1]. They can be classified into crossbar networks [2], single bus and multiple bus systems [3, 4], multistage interconnection networks [1, 5] and direct (static) networks [6].

In a direct network architecture, each node has a point-to-point connection to a set of neighboring nodes. Because nodes do not physically share memory, they must communicate by passing messages over the interconnection network. Systems used in this manner have been referred to as *message passing multicomputers.* The most commonly used direct networks are variants of the k-ary n-cube [6]. Examples of k-

ary n-cubes include the ring ($n = 1$), 2-D mesh or torus ($n = 2$), 3-D mesh or torus ($n = 3$) and hypercube ($k = 2$). These direct networks have been popular architectures for constructing massively parallel computers because they scale well; that is, as the number of nodes in the system increases, the system gains corresponding performance. Furthermore, these systems allow the exploitation of communication locality.

A major constraint with direct networks is their static topologies; once a machine is built it cannot be changed. Another problem with direct networks is that writing message passing programs has been traditionally difficult because various system primitives should be invoked to send messages between processes executing on different nodes. Shared memory programs are easier to develop than message passing programs because the programmer does not need to be concerned with the explicit movement of data. Some recent distributed shared memory designs use low dimensional direct networks. Examples of such systems include HORIZON [7], the Stanford DASH Multiprocessor [8], and the MIT Alewife machine [9].

However, whether a message passing direct network system or a distributed shared memory system is used, network latency is critical to system performance. Current second generation multicomputers are most distinguished by message routing. One method, called *wormhole routing* [10], has become quite popular in recent years. In the networks that support wormhole routing, network latency is almost independent of path length when there is no contention and message length is relatively large. Low dimensional meshes are popular topologies for such systems

because the negative effects of their large inter-node distance are minimized. In fact, most direct network topologies that use wormhole routing are low-dimensional meshes and hypercubes. The Intel Touchstone Delta [11], the Intel Paragon and the Symult 2010 [12] use a 2D mesh, and the MIT J-machine [13] and the Caltech's Mosaic [14] use a 3D mesh. The nCube-2/3 uses a hypercube topology.

To the contrary, nondirect networks such as the crossbar switch and multistage interconnection networks do not integrate processors and switches. Consequently, processors cannot communicate directly with each other. Communication between two processors in an nondirect network (as well as in multiple bus systems) is achieved through a shared memory. The logical structure of a shared memory multiprocessor allows multiple processors to access memory in a single global address space. Shared memory systems are usually considered by many to be simpler and more intuitive to program than message passing systems where one has to deal with low-level details.

Among various interconnection networks for shared memory multiprocessors, the crossbar switch network [2] gives full connectivity such that all permutations between processors and shared memory modules are possible. However, this network is unsuitable for systems with a large number of processors. The cost of an $N \times N$ crossbar switch network grows $O(N^2)$, where N is the number of processors or memory modules. The crossbar is also vulnerable to failures in communication paths because alternative paths do not exist, and so it is very often impractical.

Multistage interconnection networks [1, 5] allow a rich subset of one-to-one simultaneous connections between processors and memory modules while reducing the cost to $O(N \log N)$. The disadvantage of multistage interconnection networks is that they are not easily scalable and are subject to high communication latency. A major common disadvantage of the crossbar switch and multistage interconnection networks is that they are not inherently fault tolerant.

A multiple bus interconnection network [3, 4] is a good alternative. It provides high degree of fault tolerance, cost effectiveness, ease of broadcasting and ease of expansion. Depending on the connections of processors and memory modules to buses, different types of multiple bus systems (MBS's in short) have been introduced [18 - 23]. In the conventional MBS [4], each processor and each memory module is connected to every bus. This is usually referred to as *a full bus connection system.* This system becomes prohibitively costly when the size of the system increases. The cost of the MBS is given by $O((N + M)B)$, where N is the number of processors, M is the number of memory modules, and B is the number of buses.

Usually a bus-based system suffers from severe problems unique to the bus itself, although it has the natural advantages of a bus. Bus loading is a major problem. Because the capacitive loads and drive requirements of a bus are proportional to the number of connections to the bus, signal speed is limited by the physical capacitance associated with the bus. Thus, the maximum number of connections per bus must be chosen so as to avoid the bus loading problem. Therefore, the bus itself becomes a major limitation to system scalability.

In some cases, taking advantage of the locality of communication and/or locality of memory reference present in most applications can help in reducing the inter-node distance [17]. While many applications exhibit the locality of communication or memory reference, a conventional MBS cannot take the advantage of such locality because a bus is a mutually exclusive resource to which all processors are connected.

In this dissertation, we introduce and investigate a new bus-based system called the *Segmented Multiple Bus System* (SMBS in short) which is targeted at overcoming the above mentioned drawbacks of bus-based shared memory systems. The SMBS is targeted at exploiting reference locality, scalability and versatility, features that are not found in any of the existing bus-based systems. In the following section, we briefly review relevant literature. We describe motivation of our research and outline the dissertation in Section 1.2.

1.1 Review of Relevant Literature

1.1.1 Variants of Multiple Bus Systems

The major research effort in multiple bus systems has focused on reducing connections between memory modules and buses or between processors and buses. Several different configurations have been proposed, mostly focusing on reducing the number of bus connections while keeping the size of the MBS still moderate. Examples of such configurations include the partial multiple bus network [3], processor-oriented partial multiple bus system [15], systems with certain connection schemes like trapezoidal, rhombic, staircase, cyclic and balanced interconnection [18], partial

bus network with hierarchical request model [19], multi-level bus systems [20] and probabilistically reduced connection multiple bus system [21].

Lang, Valero and Alegre [3] proposed a bus system called *partial multiple bus system*. In the partial multiple bus network, memory modules are divided into equal sized groups and each of these groups is connected to some equal numbered, but different subset of buses. The processors are connected to all the buses. This partial multiple bus structure was intended to reduce the connection cost while trading off an acceptable degree of system bandwidth. They reported a decrease in system bandwidth of at most 6% while reducing memory-bus connections by half compared to the conventional full bus connection system.

In the processor-oriented partial multiple bus structure [15], bus connection complexity is the same as that of the partial multiple bus network. In that system, processors are partitioned into equal sized groups and are connected to different subsets of buses, while memory modules are connected to all the buses. With improved load balancing in the arbitration mechanism (a memory module that has outstanding requests must be granted an available free bus), the system is claimed to have an increase of up to 20% in system bandwidth over the partial multiple bus system [3]. However, in the processor-oriented multiple bus system, implementation of the load balancing arbiter is very complicated.

Lang, Valero and Foil [18] proposed some connection schemes like rhombic, balanced, staircase, cyclic, etc.. In these systems, not all the buses are connected to all the memory modules; however, any B memory modules can be connected to B

buses by some connection scheme and all the buses have full processor connection. Therefore, the throughput is the same as that of a full bus connection system. However, the implementation of arbitration algorithms is complex and the connection cost of a large system is still high.

The partial bus network with hierarchical request model [19] divides memory modules into a few classes and connects more frequently addressed memory module classes to more buses than those that are less frequently referenced. Processors are connected to all the buses in this system as well. Even though system bandwidth is comparable to that of partial connection multiple bus systems, the connection cost is higher. Because of non-uniformity of connection, fault tolerance could be serious for some memory module classes and implementation of arbiters would be complex.

In the multi-level bus system [20], processors and memory modules are divided into a number of multi-level clusters. As the cluster level goes up, connectivity of a bus increases and finally at the highest level, a bus has full connectivity. The total number of connections is less than that of the partial multiple bus system [3] with same number of processors. The multi-level bus system becomes cost effective only for hierarchical reference models. In other words, the multi-level bus system will not perform well under a uniform request model.

Karim [21] proposed a probabilistically reduced connection multiple bus system. His approach was motivated by the hypothesis that there are many connections in conventional multiple bus systems to satisfy highly improbable request patterns; in other words, probabilistically redundant (or under-utilized) connectivity exists. His

results show that the typical memory-bus connectivity cost reduction is in the range of 20% – 37%, depending on the assumed request model. He used several request models, such as the uniform request model, hot spot request model, locality-based request model and locality-based request model with local hot spots [21]. However, the proposed system also has full connectivity between processors and buses. The author reported another approach to reduce the connectivity cost from both processor or memory sides which did not show any significant cost improvement.

Basically, all of these partially connected systems solve the bus loading problem to some degree. They achieve some reduction in connection cost while trading off an acceptable and tolerable degree of performance degradation. However, these architectures have full connectivity in the sense that either every processor is connected to every bus or every memory module is connected to every bus. This full connectivity between processors and buses or memory modules and buses gives rise to serious engineering difficulties as system size increases. They also increase the cost to an unacceptable level. Among these difficulties, the problem of bus loading is a major one.

To solve the bus loading problem, a new class of multiple bus architectures, called ***binomial multi-bus architecture*** based on binary codes, was proposed [22]. This class is aimed at further reduction in the interconnection complexity. It was shown in [22] that the reliability of this architecture class is as good as that of a full bus connection system while the number of bus connections is almost half of that required in the full bus connection case.

To interconnect a large number of processors using a minimum number of buses when both the number of input and output ports on a processor and the bus fanout are constrained, general construction techniques of systems based on *Balanced Incomplete Block Design* were introduced [23]. While these multibus architectures are aimed at further reduction in the number of connections to buses, for truly large systems the fanout constraint cannot be overcome by resorting to these partial configurations.

1.1.2 Overview of Wormhole Routing

The performance of a direct interconnection network is greatly dependent on the performance of its communication network; so an efficient routing algorithm is critical to the performance. Routing can be classified as *deterministic* or *adaptive*. Deterministic routing completely determines the path only by the source and destination addresses. If routing is adaptive, the path for a given source and destination pair is determined based on dynamic network conditions, such as failure of channels or blocked channels.

In order to obtain good performance, it is desirable that a routing algorithm uses a shortest possible path for each packet. Such a routing algorithm is said to be *minimal*. A *non-minimal* routing algorithm may use a longer path for each packet, usually in response to dynamic network conditions. Non-minimal routing should be used carefully so as to avoid *livelock* situations. Livelock can occur when routing is continued indefinitely without finding the destination.

Routing algorithms can be further classified by the type of switching technique they utilize. In *store-and-forward* (or *packet*) *switching*, whenever a packet arrives at an intermediate node, the entire packet is stored in a packet buffer. Then the packet advances to the next node when the next channel is available and the node has available packet buffer. In *circuit switching*, a physical circuit is established exclusively for a message between the source and destination nodes. After the message has been transmitted along the circuit to the destination, the circuit is released. Therefore, there is no need to have buffer space at each node.

In *virtual cut-through* [30], the packet header (which has routing information) is examined upon arrival at an intermediate node. The packet is buffered at the intermediate node only if the next channel is blocked. Otherwise, the packet advances immediately without buffering. Both circuit switching and virtual cut-through are based on the concept of cut-through, which can significantly reduce network latency.

Although both virtual cut-through and circuit switching offer significantly reduced network latencies when there is no contention, virtual cut-through requires large buffers to store blocked packets and in circuit switching the established physical circuit between the source and destination nodes cannot be shared among other messages.

Wormhole routing [10] which has become quite popular in recent years, resolves these drawbacks while offering similar low network latency. A packet is divided into a number of flits (*flow control digits*) for transmission. The flit at the head of the

packet, the *header flit*, has routing information. As the header flit advances along the path, if there are no blocked channels, the remaining flits follow in a pipelined fashion. If the header flit is blocked at some intermediate node, the trailing flits are blocked in place, with several channels being occupied simultaneously until the header flit can move forward. The channel once occupied by a packet will only be released by the tail flit. The pipelining property of wormhole routing could make the network latency largely insensitive to path length if there is no contention. Another attraction of this routing technique is that each node requires only a small first-in first-out (FIFO in short) flit buffer to store a few flits for each channel. Low dimensional meshes and hypercubes are the most popular direct network topologies used in wormhole routed systems because the negative effects of their large inter-node distance are minimized.

If a packet is blocked at every intermediate node, virtual cut-through behaves exactly the same as packet switching. On the other hand, if all of the intermediate channels are free, virtual cut-through becomes very similar to circuit switching. Hence, cut-through switching is advantageous when the network load is low. In the ideal case, when flit buffer capacity is unbounded, wormhole routing is equivalent to an optimized form of virtual cut-through. In that case, wormhole routing allows a partially buffered packet to be forwarded as soon as its outgoing channel becomes available.

A situation that can postpone packet delivery forever is called *deadlock*. Deadlock can occur if a circular wait condition happens; a packet may wait for a network

resource while holding other resources and excluding other packets from acquiring the held resources. In packet switching and virtual cut-through switching, the deadlock resources are buffers. Therefore, they need extra buffers for deadlock-free routing. In circuit switching and wormhole routing, the resources are channels. Because blocked packets hold several channels and their corresponding flit buffers simultaneously, wormhole routing is particularly susceptible to deadlock. *Starvation* is similar to deadlock, but occurs when a packet waits for an event that can happen but never does. For example, a packet may wait forever to acquire a network resource for which other packets are always competing successfully. Starvation is only possible when the allocation of network resources is unfair.

Generally methods for avoiding deadlock routing use one of two approaches: dimension ordering [6, 11, 12] and virtual channels [31]. Dimension ordering [6, 11, 12] routes packets along the different dimensions in a pre-specified order. One dimension ordering approach, LR routing [6] (also called *e-cube* routing algorithm), applied to hypercubes, routes a packet by selecting intermediate nodes such that the address bits gradually match from the most significant bit to the least significant bit (or left to right). The xy routing algorithm [11, 12], applied to two-dimensional meshes, routes a packet first along the x dimension and then along the y dimension. These dimension ordering approaches guarantee deadlock-free routing but are non-adaptive.

In the other approach [31], a single physical channel is divided into multiple disjoint virtual channels and a cyclic dependency-free routing is employed. Generally each virtual channel has reduced bandwidth because virtual channels must share

one physical channel. However, virtual channels make adaptive routing possible and produce flexible flow control.

1.1.3 Review of Relevant Performance Analysis

In this subsection, we review the work of several researchers on performance analysis of asynchronous MBS's and direct networks with wormhole routing which relate to our work.

Jackson [44] showed that if a queueing network consists of exponential queues, its solution is separable. The term 'separable' comes from the fact that each service center can be separated from the rest of the network and its solution is evaluated in isolation. Then, the solution of the entire network can be formed by combining these separate solutions. This simplifies the evaluation while ensuring accuracy.

For circuit switched MBS's, the path between a requesting processor and a memory module is established during the entire memory operation. This simultaneous possession of a bus and a memory module represents a non-separable aspect which results in an approximate bus-memory system.

Asynchronous operation has been studied using Markovian queueing network models for MBS's [33, 34]. Marsan and Gerla [33] gave an approximate solution to a Markovian model developed under the assumption that the memory service time and request interval are exponentially distributed. In their solution, all memory modules and processors in the system are assumed to be identical and a uniform memory reference model is assumed in order to reduce the size of Markov chains.

Irani and Onyuksel [34] presented a closed-form solution for a Markovian queueing model. Their analytical results are claimed to be easily applicable to a system of any size.

Towsley [35] developed two classes of approximate models for asynchronous MBS's. These are the combined bus-memory model based on the *flow equivalence* technique and the *surrogate delays model*, in which the mean-bus queueing delay is added to the mean-memory holding time.

Another approximate simple queueing model of asynchronous MBS's was presented by Yang and Zaky [36]. In this model each processor has a local memory and thus is able to continue processing while waiting for response from the shared memory. All these previous works on MBS's have focused only on circuit switched systems [33 – 36].

For packet switched MBS's, there are three different types of service centers, typically: processor, bus and memory service centers. The solution is separable if these three service centers have exponential queues with exponential distribution of service times. The bus group in a packet switched MBS can be modeled as a single service center called *flow equivalent service center* (FESC) [16]. The bus group is called the *aggregate* and the remaining service centers in the system are called collectively the *complement*. From the point of view of the complement, an FESC is a single service center whose behavior is identical to the aggregate itself. Consequently, the purpose of the FESC is to represent the behavior of the aggregate numerically.

FESCs are represented in queueing network models using *load dependent service centers.* A load dependent service center can be considered as a service center whose service rate is determined by a function of the number of customers in its queue. As viewed by the complement, this behavior corresponds to the flow of customers out of the aggregate and into the complement. An approximation for this flow can be obtained under the assumption that the average service rate at the aggregate depends only on the state of the aggregate which is defined by the number of customers in the aggregate [16].

Yang and Bhuyan [37] presented approximate queueing network models for both packet switched synchronous and asynchronous MBS's. Basically, all processors, memory modules and buses are assumed to be identical and typically a uniform memory reference model is assumed in [37]. For the synchronous system, the analysis is based on the assumption that each of the shared resources in the system is represented as a single server queue. For instance, each processor has a bus access queue and every memory module has both an input and an output queue.

For the asynchronous system, the centrally controlled bus system is modeled as an FESC. For decentralized control of the bus system, each bus is assumed to work independently of others. For both bus control strategies (the centralized and the decentralized), memory access time and packet transmission time are assumed to be deterministic.

Dally [38] analyzed k-ary n-cube networks of varying dimensions under the assumption of a constant wire bisection width constraint, which is motivated by wiring

density limitation in VLSI. He has shown that low-dimensional networks (e.g., tori) have lower latency and higher hot spot throughput than high-dimensional networks (e.g., binary n-cube) with the same bisection width.

Agarwal [39] derived simple closed-form equations taking both switch and wire delays into account. The model includes the effects of packet size and communication locality. Under the constraint of constant wire density and constant bisection width, a two-dimensional mesh yields the lowest latency. However, when node delays are taken into account, it is shown that the best network would have three or four dimensions. Longer messages make the relative effect of network distance less important, while shorter message lengths increase the relative influence of network distance and tend to favor networks with more dimensions. Communication locality enhances the attractiveness of low-dimensional networks.

Scott and Goodman [40] examined the performance impact of pipelined channels on k-ary n-cube networks on the choice of dimensionality and radix. Networks are investigated under the constant link width, constant node size and constant bisection width constraints. Pipelined channel networks have higher optimal dimensionality than non-pipelined channel networks. Their radixes remain roughly constant as network size grows, decreasing slightly for some unidirectional tori and increasing slightly for some bidirectional meshes. In [38 – 40], studies of k-ary n-cubes with wormhole routing have shown that under reasonable assumptions, the optimal dimensionality is 2 to 4.

Adve and Vernon [41] developed approximate MVA models for 2-D mesh interconnection networks with wormhole routing. The model explicitly represents the virtual channels used in the non-adaptive deadlock-free routing scheme developed by Dally and Seitz [31]. Results presented in [41] showed that when processors have multiple outstanding requests, the system does not scale well with increasing system size because of bandwidth limitation. However, for a very high level of communication locality, the system scales well. For the case of four outstanding requests, for example, at least $70\% - 80\%$ of each processor's requests must be directed to its nearest neighbors. This may correspond to unrealistic workloads.

Kim and Das [42] modeled a deadlock-free routing scheme in asynchronous hypercubes for finding the average delay of a message in the network. Their model can capture the probability of blocking caused by the LR wormhole routing scheme and the random wormhole routing scheme. They modeled a set of blocked channels as an M/M/1 queue. They claimed that their analytical results show more close agreement with simulation results than analysis of wormhole routing first reported by Dally [38] and that the model can capture any message destination distribution.

Hybrid switching proposed by Shin and Daniel [43] dynamically combines features of both virtual cut-through and wormhole switching by buffering a small fraction of blocked packets and limiting the number of links that blocked packets can hold. This significantly reduces the buffer requirement at intermediate nodes compared to virtual cut-through and provides higher throughput than that of a traditional wormhole routing network.

1.2 Motivation and Outline of Dissertation

A typical bus-based system suffers from severe problems unique to the bus itself. First, the bus is a mutually exclusive resource. Second, in MIMD systems, all processors must participate in an arbitration phase. Third, the bus clock cycle must be long enough so that signals can propagate through the entire length of the bus. Fourth, the capacitive loads and drive requirements of a bus are proportional to the number of connections to the bus. Fifth, signal speed is limited by the physical capacitance associated with the bus.

Because of these drawbacks, the number of connections to a bus has to be limited to a few dozens. Hence, the first objective of our research is to overcome the architectural limitations of bus-based shared memory systems while maintaining their advantages in terms of high degree of fault tolerance, ease of expansion and ease of programming.

To avoid bus loading problems in large systems it is essential to allow only a certain small number of connections to a bus. Segmenting a large bus system into a number of smaller ones can be an attractive option. A large multiple bus system could be segmented into smaller partitions, which we call *segments*, if these segments can be connected somehow such that bus loading effects are avoided. This makes it possible to overcome most problems associated with traditional bus-based systems. Each segment is in effect a multiple bus system whose size is chosen so as to avoid bus loading problems. The short bus results in removing bus loading problems and permits maintaining a fast bus cycle.

In a segmented system, several processors may access different memory modules simultaneously, if they do not require an overlapping set of memory modules. For instance, this would be the case if processors access memory modules within their own segments. This will tend to increase the effective bandwidth of the segmented system over that of a non-segmented bus system at a lower cost. This is the concept underlying this dissertation.

In applications running with *uniform memory access* running on a typical multiple bus system, latency for all communications increases as system size scales. Hence, mechanisms used to implement uniform memory access networks are not scalable. The performance of those networks is quite sensitive to *memory access latency*, which is the delay from the time a processor requests a memory access to the time when data is returned from the memory module (or is stored in the module). Unless a mechanism that is less sensitive to the memory access latency is developed, scalability of the system will be hampered.

The hierarchical request model proposed by Chen and Sheu [19] improves the partial multiple bus network by connecting more frequently addressed memory modules to more buses than those that are less frequently referenced. For the multi-level bus system proposed by S. Mahmud [20], it has been shown that a processor accesses its nearest memory modules more frequently than others. However, these two networks have two major drawbacks:

1. The connection complexity of certain buses is the same as that of a conventional MBS.

2. The connection patterns lack regularity.

Both studies described above are done in the context of applications with non-uniform request patterns running on systems which can best support uniform memory access.

In contrast to these networks, the segmented system we propose in this work can be classified as a non-uniform memory access network although each segment has regular connections. With such a non-uniform memory access network, applications where memory reference patterns tend to favor near memory modules can run efficiently. Therefore, the segmented system can utilize memory reference locality which is usually not the case in conventional bus-based systems.

This dissertation introduces a new class of systems called *Segmented Multiple Bus System* (SMBS) based on the fundamental ideas described above. The SMBS targets multiprocessor systems with increased scalability while keeping the natural advantages of a bus-based system. The SMBS provides a global address space. All processors can transparently access all memory locations. This makes programming intuitive and easy. The SMBS exploits the natural advantages of the bus by using relatively small bus segments, here called *segment buses.*

One of the unique characteristics of the SMBS is that it allows the exploitation of communication locality even though it is a bus-based nondirect network. By exploiting the locality of memory reference inherent to many applications, scalability of the SMBS could be drastically improved. Another interesting feature of the SMBS is that it supports wormhole routing which is mostly used in direct net-

work topologies. Together, both segmentation and wormhole routing could make the system largely scalable even though it is a bus-based system. This is a major hypothesis underlying the work we report here.

Another important aspect of this research is that we develop novel performance models for wormhole routed SMBS's. This, to our knowledge, is the first attempt to adapt wormhole routing to a bus-based (non-direct) network. We will therefore incorporate features of both direct and nondirect networks in our performance modeling. Among direct network features we include are the effects of blocking and that of the pipelining property of wormhole routing. The bus system in each segment is modeled as *a flow equivalent service center* (FESC) [16] representing multiple servers with a single central queue. We should note that this central server representing the bus system is a load dependent server. This is a unique modeling feature for multiple bus systems which we will encounter frequently.

In our work we conjecture that packets quickly delivered to their destination will reduce the contention between other packets for network resources and that simultaneous communication traffic can thus be reduced. For that reason, a transient packet which passes between segments will be given priority over others in the bus assignment process.

The models we derive are closed multi-class queueing network models. For analysis of this type of closed queueing network, with features that violate separable model assumptions, we use *Approximate Mean Value Analysis* [16]. Our models are very complex and feature aspects of both direct and nondirect networks with non-

product form queueing behavior. Development of such models to reflect network behavior accurately is a major contribution of this dissertation.

The rest of this dissertation is organized as follows. Chapter 2 introduces the Segmented Multiple Bus System (SMBS) and discusses its distinguishing features. Architectural features of the SMBS such as its interconnection structure and arbitration mechanism are described. In Chapter 3, we develop a performance model of the SMBS with single flit buffers for finding the mean response time of a request issued to a shared memory module in an asynchronous environment. We state the assumptions regarding system workload and operation, and describe key performance issues. In the model, representation of the packet pipelining property of wormhole routing and blocking caused by finite flit buffers are considered. In Chapter 4, we develop a performance model for the SMBS with infinite flit buffers. When buffer capacity is unlimited, wormhole routing is equivalent to an optimized virtual cut-through scheme [30]. The infinite flit buffer model does not consider the mean residual residence time of a tail flit and blocking delay because buffer space at a segment switch is unlimited.

Chapter 5 describes our event-driven simulator, presents simulation results to validate our analytical models and presents the analytical results for our performance analysis. By comparing system performance with the single flit buffer model against that of the infinite flit buffer model, we shall investigate how much system performance can be enhanced by increasing flit buffer sizes. We study the performance under a uniform memory reference pattern and then examine the impact

of memory reference locality on the mean response time and scalability. We also address how performance changes by varying the number of segment buses and the number of flits per packet. Given a fixed sized packet, the number of flits per packet is dependent on the bus width which is a system design parameter. Our results will show that network latency is more influenced by bus contention than network distance. Thus, a small number of flits per packet and a wide bus width should help to reduce the probability of contention and consequently the expected waiting time for a segment bus. Chapter 6 summarizes the research reported in this dissertation and presents ideas for future research.

CHAPTER 2

SEGMENTED MULTIPLE BUS SYSTEMS

In this chapter, the architecture of the Segmented Multiple Bus System (SMBS) along with its distinguishing features are introduced. Architectural features of the SMBS such as its interconnection and arbitration mechanism are described.

2.1 PRELIMINARIES

The SMBS may be classified as a general purpose asynchronous MIMD machine with shared memory. The SMBS consists of N processors and M shared memory modules which are evenly divided into g identical segments numbered from 0 to $g-1$. Each segment is thus a relatively small multiple bus system that is composed of $n = N/g$ processors, $m = M/g$ memory modules, and b segment buses. Here we shall refer to a bus in a segment as a *segment bus*. The segment bus is bidirectional.

In the SMBS we will use wormhole switching [10] (better known as wormhole routing). At both ends of a segment bus we utilize *segment switches*. The segment switch is unidirectional and is a switching device with its own flit buffer. Although the segment bus is bidirectional, the bus behaves as a unidirectional bus when it is involved in an inter-segment access in which a memory request (or a reply) may be transferred to a neighboring segment through a unidirectional segment switch. The segment switch supports wormhole switching and has intelligence so that it could handle memory requests across segments. The segment switch checks the destination

address of a memory request (or reply) and forwards the request (or reply) to the appropriate memory module (or processor). Aside from routing functions, segment switches also serve the function of isolating (or buffering) adjacent segments. Thus, in SMBS's, bus loading is not an issue if segments are not too large. In each segment, the switches are divided into two groups of equal size but opposite direction . We call the flit buffers in the first group of segment switches *upstream flit buffers* and the flit buffers in the second group *downstream flit buffers*, based on the direction of flow they support.

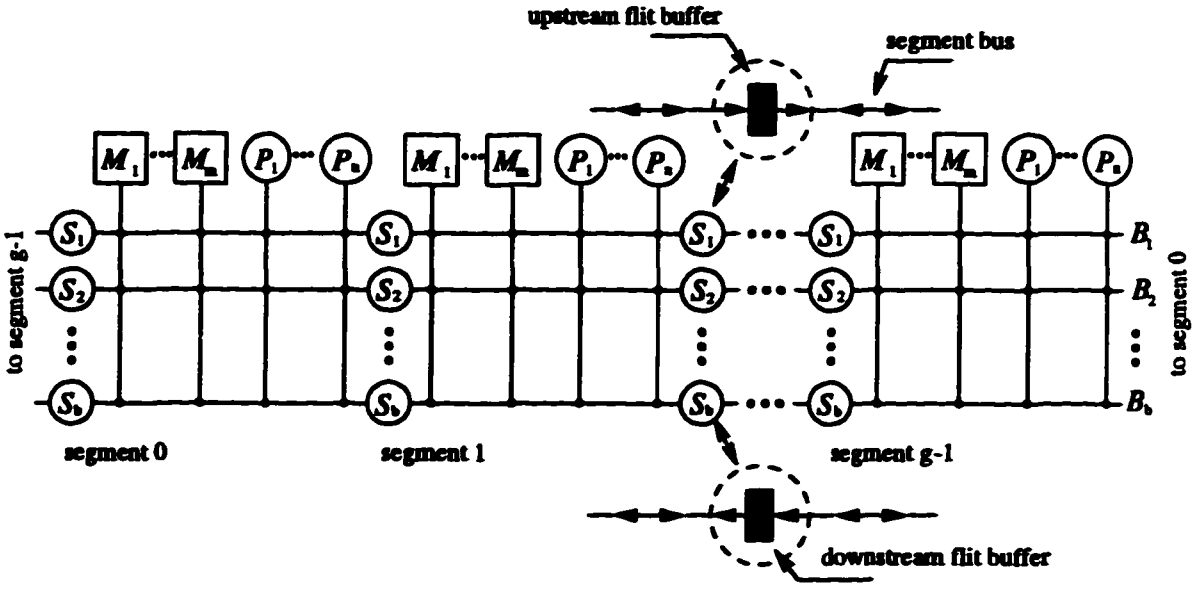


Figure 2.1: Segmented multiple bus system.

Figure 2.1 illustrates the interconnection of the SMBS, where each segment is assumed to have a full bus connection.

We define *a local memory module* as a memory module directly accessible by a segment switch to which the memory module is connected by a segment bus. Other

memory modules are referred to as *non-local* with respect to the given segment switch. From a processor's point of view, memory modules connected by the segment bus that the processor is connected to, are called *local* and the other memory modules are referred to as *non-local*.

The *bus cycle* is also considered the *flit time*. It is equal to the time needed to take a flit from one switch to the next over a segment bus or to a local memory module. Passing flits between two adjacent segment switches uses a handshaking protocol similar to the one described in [32]. A dedicated single-bit Request/Acknowledge line is associated with two adjacent segment switches. A flit is moved to the adjacent flit buffer only if it is empty.

To reserve bus bandwidth, only those buses that have a non-empty flit buffer at the sending side and an empty flit buffer at the receiving end may participate in the bus assignment process in any given cycle. Each memory module has an input buffer and an output buffer. This makes it possible for memory modules to serve requests continuously without having to wait to transmit the response packet back to the requesting processor before serving other requests.

When a memory request is issued by a processor, the request will traverse segment switches if the requested memory module is non-local or will be sent directly to the memory module if the module is local. To make such determination, each segment switch should "listen" to the segment bus. However, this may cause the segment switch to suffer the unnecessary overhead of decoding every request. In order to resolve this problem, we assume that the request has one extra bit which

indicates whether it is an intra-segment (local) request or an inter-segment (non-local) request. This eliminates unnecessary buffering of the request at a segment switch.

The connection topology of the system could be a linear connection of g segments or a ring with end-around connections. In a ring connection, each processor may reach any distributed shared memory module by traversing at most $\lfloor \frac{g}{2} \rfloor$ intermediate segment switches. The physical symmetry of the system can lead to a balanced utilization of segment buses. When segments are connected in a ring topology, traveling along one direction may be shorter than the other. However, because segments are connected in a ring-like topology, deadlock may occur when wormhole routing is employed. Deadlock comes from a circular waiting for network resources as we discussed in Chapter 1. We can simply solve this deadlock problem by unbounding the size of flit buffers. We consider only the case of infinite flit buffers in a ring-like connection topology. However, the SMBS connected in a linear fashion would be deadlock-free even when the size of a flit buffer is finite.

The arbiter of the system is naturally asynchronous because the SMBS will operate in an asynchronous MIMD mode. While an asynchronous arbiter is more suitable for the system, it is not reliable because inputs may change even if the arbiter is not in a stable state. Designs for reliable asynchronous arbiters have been developed and synchronization of asynchronous inputs has been studied. However, the synchronization process is still not perfect [25 – 28]. Details on arbiter

reliability is out of the scope of this work. In the next section, we will discuss the arbitration scheme of the SMBS in detail.

The SMBS is of interest for several reasons. By distributing the shared memory modules among the segments, memory reference locality can be exploited to reduce network traffic and network latency. Further, the segmented bus structure resolves the bus loading problem associated with traditional multiple bus systems. Because segment buses can be made short with limited number of connections, data transfer over them can be very fast. Moreover, the overall bandwidth of the SMBS increases proportionally with the number of segments although increasing the number of segments also increases latency by a corresponding amount. This architecture still keeps many of the advantages of a traditional multiple bus system like shared memory (logically), high degree of fault tolerance, etc.. An SMBS which can support a very large number of processors, can utilize wormhole routing which is less sensitive to an increase in the number of segments. This results in a system with increased scalability. We will elaborate on this aspect later in Chapter 5.

2.2 Arbitration Mechanisms

2.2.1 Parallel Arbitration in Multiple Bus Systems

One of the most important design aspects of a multiple bus multiprocessor system is the arbitration mechanism, which provides the control function for the system. We assume that all memory modules are single-port memories, so that at most one access can be made to a module at a time. The conventional connection in a traditional multiple bus system (MBS) calls for all N processors and M memory modules to

be connected to all B buses. For the MBS , circuit switching is usually assumed. Thus, the path is set up until a response message returns. For such a system the arbitration system could carry out the following functions in parallel [24]:

- Processor selection when simultaneous multiple requests for a shared memory module occur.

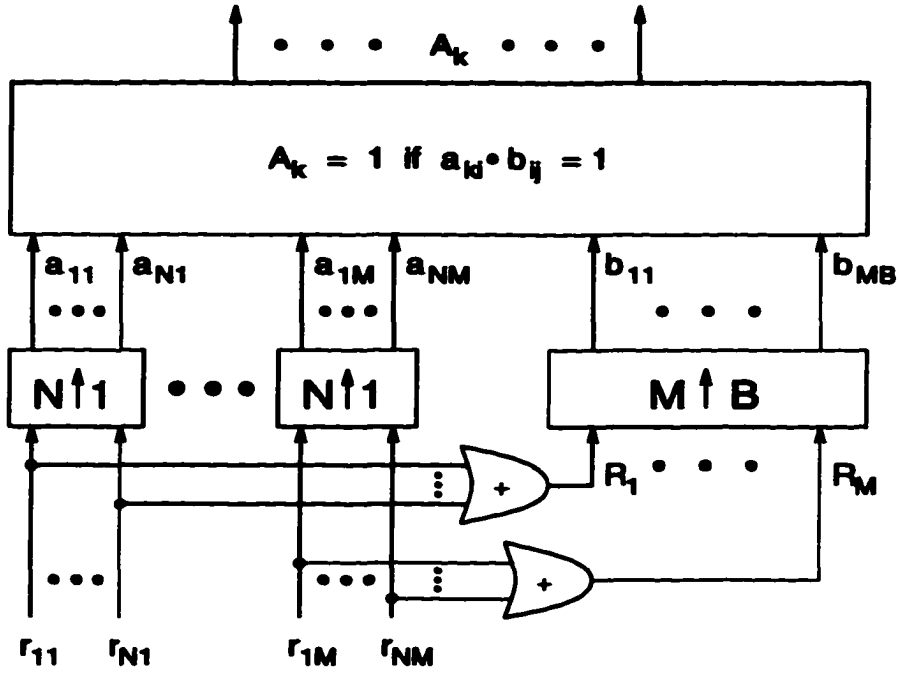
  This is solved by using an N-user 1-server arbiter, denoted as $N \uparrow 1$. The number of such arbiters functioning in parallel is M.

- Assignment of buses from existing free buses to shared memory module requests.

  It is necessary to assign b buses ($b \leq B$) to i requests ($i \leq M$). Each request corresponds to a request made by a processor to a given memory module.

The general structure of the arbitration system in a parallel arbiter implementation is shown in Figure 2.2. Processor selection and bus assignment are carried out in parallel in order to reduce arbitration and assignment delay.

The processor selection function is implemented using as many $N \uparrow 1$ arbiters as memory modules. The responses of the $N \uparrow 1$ arbiters, together with the output of the $M \uparrow B$ arbiter, make it possible to obtain the final grant signals in parallel. Processor k requesting memory module i ($r_{ki} = 1$), obtains the final grant signal ($A_k = 1$) if the arbiter for memory module i selects processor k ($a_{ki} = 1$) and memory module i is assigned to bus j ($b_{ij} = 1$ for some j) in the bus arbiter.



N↑1 : N-user 1-server memory arbiter
M↑B : M-user B-server bus arbiter
r_{kd} : Request from processor k to memory module i
R_i : Request for memory module i
a_{kd} : Memory module i assigned to processor k request
b_{ij} : Memory i access by bus j
A_k : Grant (memory module and bus) to processor k

Figure 2.2: Arbitration system structure of an MBS.

2.2.2 Arbitration Mechanism for the SMBS

The SMBS is a cascaded connection of g identical segments. Each segment is similar to a conventional multiple bus system except that these segments are connected via a set of segment switches. A memory request from a processor may be directed toward a local memory module or it may be transfered to a neighboring segment through a segment switch. The former is called *an intra-segment access* and the latter is called *an inter-segment access.* In the case of an inter-segment access, the packet for a memory request traverses (possibly several) segment switches and is eventually sent to the target memory module. A response packet from the memory module takes a return path which could be different from the forward path. We assume that a transient packet at a segment switch has priority over other packets in bus assignment. This is intended to reduce network contention.

Instead of using a central arbitration system, which would hamper the scalability goal for the SMBS, we employ a distributed arbitration mechanism. Each segment has an identical arbitration system and arbitration in each segment is independent of those in other segments. Hence, we can consider arbitration within a segment only.

Resources within a segment consist of n processors, m shared memory modules, b segment buses, and a total of $2b$ segment switches which are located on the boundary of the segment. Notice that there are b segment switches on each side of the segment ($2b$ total) but only half of those can access segment buses and memory modules in the segment. For simplicity, we assume that the number of buses in a

segment is even. The segment buses are divided into two groups: *an upper half bus group* and *a lower half bus group*. Each bus group is associated with a bus arbiter. The upper bus group (lower bus group) consists of the first half (second half) of segment buses which are bidirectional. The segment switches which are unidirectional are also divided into two groups of equal size but opposite direction.

Two sources of conflict exist, which are the same as those in a conventional MBS. First, more than one request can be made to the same memory module. Second, available buses may not be enough to accommodate all the requests. As in a conventional MBS, the arbitration scheme in a segment employs a processor selection function and a bus assignment function. These two functions can be implemented partially in parallel, like those in an MBS, as we explain next.

In each segment, a two-phase arbitration scheme is employed to resolve the conflicts mentioned above. In the first phase, a segment bus is assigned to a segment switch if a request exists at the switch. The request could be caused by a transient packet or a terminating one. In the second phase, both processor selection and bus assignment are needed. In the case of an intra-segment access, these two functions can be implemented in parallel as would be the case in an MBS. Processor selection is needed when simultaneous multiple requests for a shared memory module occur. In this phase at most b outstanding requests from segment switches (selected in the first phase) and at most n requests from local processors can coexist. Contention for a given memory module is solved by using an $(n+b)$-user 1-server arbiter. Thus, a total of m such arbiters is needed per segment. If the arbiter selects a request from

a segment switch that passed the first phase, the request is immediately assigned to the memory module. If the selected memory request is from a processor, that request must obtain the final grant from a bus arbiter in the same way as in the parallel arbitration of the MBS discussed earlier.

Together with memory module arbitration, bus assignment for free buses is employed in the second phase. Each segment has two $(2m+n)$-user $b/2$-server arbiters for bus assignment: one for the upper bus group and one for the lower bus group. Each such arbiter assigns at most $b/2$ buses by selecting requests from $(2m+n)$ request inputs. Among the $2m$ requests, up to m requests could come from memory response packets and up to m requests could come from outstanding memory requests selected by the memory arbiters. Notice that in each segment, there may be $(2m+n)$ different request inputs for bus assignment: n for inter-segment accesses by processors, m for requests from memory output queues for reply packets, and m for outstanding memory requests from intra-segment accesses. However, notice that the maximum number of requests that could actually participate in the bus assignment part of the second phase is $(m+n)$ because the n inter-segment accesses from processors and the m outstanding memory requests from intra-segment accesses must originate at the same set of n processors.

The bus assignment process is designed so as to immediately assign a bus (if available) to a memory module with a response packet or to a processor requesting an inter-segment access. If a bus is assigned to an outstanding memory request, however, this bus assignment, together with the output of the corresponding

$(n+b)\uparrow 1$ memory arbiter, is used to produce the final grant. Thus, arbitration for an intra-segment access is done in parallel as in the traditional MBS. The complete arbitration system that each segment of the SMBS needs to have is built with m arbiters of the $(n+b)$-user 1-server type and two arbiters of the $(2m+n)$-user $b/2$-server type.

Even though the arbitration system carries out processor selection and bus assignment in parallel, it is critical that each arbiter be fast, efficient, simple and of modular structure. For our system we need two types of arbiters which we can generically denote as: $N\uparrow 1$ and $M\uparrow B$.

Pearce, Field and Little [25] presented an implementation of an asynchronous $2\uparrow 1$ arbiter module suitable for implementing N-input arbiter trees. An $N\uparrow 1$ arbiter can be constructed from $2\uparrow 1$ arbiter modules with an arbiter tree of depth $\log_2 N$. This arbiter tree is fast because the arbitration time grows as $O(\log_2 N)$. When each $2\uparrow 1$ arbiter receives simultaneous requests the arbiter alternates priorities between the two inputs to maintain fairness of arbitration. When a multi-input arbiter tree is constructed from the $2\uparrow 1$ arbiter modules presented in [25], the delay of each level would be 3Δ, where Δ denotes the nominal gate delay. Thus, the total delay for an $N\uparrow 1$ arbiter tree is $3\Delta \log_2 N$. Therefore, the corresponding delay of an $(n+b)\uparrow 1$ arbiter tree for our arbitration system is $3\Delta \log_2(n+b)$.

Lang and Valero [29] presented implementations of $M\uparrow B$ arbiters. The $M\uparrow B$ arbiter consists of M combinational arbiter modules that produce the B bus assignments. The arbiter is cyclic in order to obtain a fair policy and the delay is

proportional to M, the number of arbiter modules. Hence, the arbiter design is simple but relatively slow. If each combinational module can be implemented by a programmable logic array (PLA), with a delay of 3Δ, the total delay of the arbiter would be about $3\Delta M$. For our arbitration system, we can approximate the corresponding delay of a $(2m+n)\uparrow\frac{b}{2}$ arbiter to be $3\Delta(2m+n)$.

Turning to the performance of the arbitration system we just described for the SMBS, we observe that the total delay of the arbitration system is the sum of the delays for the two phases. Delay for the first phase of the arbitration can be ignored because the time for a bus arbiter to assign a segment bus to a segment switch (if a request exists at the switch) is relatively small. The second phase of the arbitration process consists of two parallel functions: processor selection and bus assignment. Based on the above, the delay for the processor selection function which is implemented by $(n+b)\uparrow 1$ arbiters is $3\Delta\log_2(n+b)$ and the arbitration delay of a $(2m+n)\uparrow\frac{b}{2}$ arbiter, used for bus assignment, is $3\Delta(2m+n)$.

There are two different cases for arbitration in the second phase:

1. A processor selection and a bus assignment functions in the case of an intra-segment access, which could be implemented in parallel; and

2. A bus assignment function only, in the case of an inter-segment access.

Hence, the worst delay of the second phase arbitration is given by

$$max(3\Delta\log_2(n+b), 3\Delta(2m+n)).$$

If b is smaller than n or m, the delay would be $3\Delta(2m+n)$. Thus, the arbitration delay in each segment becomes equal to the delay of the $(2m+n)\uparrow\frac{b}{2}$ arbiter.

In conclusion, the arbitration delay in the SMBS is dependent not on system size but on the size of a segment because arbitration at each segment works independently. Arbitration delay is not a function of the number of segments the system consists of. This is a very attractive feature of the SMBS.

2.3 Scalability of the SMBS

Each segment can be viewed as a conventional MBS. Thus, constraints on bus loading apply to a segment in exactly the same way it applies to an MBS. However, once each segment is expanded to the allowed maximum number of connections per bus such that a given speed is reliably maintained, we can increase system size by taking advantage of the topological characteristics of the SMBS. The system is a cascaded connection of segments. For further scale-up of an SMBS with a linear connection, segments can be added at the ends. In a ring connection of segments, the ring can be opened and additional segments can be inserted into the ring.

An SMBS which supports wormhole routing is likely to be less sensitive to the increase in the number of segments. Unlike a conventional MBS, our proposed segmented bus system can take advantage of the locality of memory reference. This gives the designer more options when it comes to system scalability. Scalability performance issues will be discussed later in Chapter 5.

CHAPTER 3

QUEUEING ANALYSIS FOR SYSTEMS WITH SINGLE FLIT BUFFERS

This chapter develops a performance analysis model for studying SMBS's with single flit buffers employed at each segment switch. We will first state all modeling assumptions and discuss input model parameter selection in detail. Using approximate Mean Value Analysis we then develop equations for mean residence time a packet is expected to incur at each queueing network service center. Residence times at three distinct service centers are evaluated; namely we will evaluate residence times at a segment switch, a processor and a memory module.

3.1 ASSUMPTIONS AND NOTATION

Two types of messages are generated in the shared memory SMBS: a request and a reply. A request message represents a memory *read* or a memory *write* request. A reply message is either an *acknowledgment* message or a *data* reply message. A data reply message carries data sent back to a processor from a memory module in response to an earlier read request. For simplicity, each message is assumed to be a single packet of a constant size, although our analysis can be extended to the case of different request and reply sizes. The read and write requests are assumed to be treated equally in the sense that a processor will remain idle after submitting a request until it receives either data for a read request or an acknowledgment for

a write request. Thus, the total number of packets in the system at any time is constant and is equal to the number of processors in the system. Hence, the entire system behaves like a closed queueing network.

The model we develop is general enough so as to allow analysis of systems with uniform as well as non-uniform memory reference patterns. Memory requests within a segment are assumed to be uniformly distributed. However, at the segment level, references are assumed to be distributed arbitrarily. Each memory module is assumed to have an input buffer and an output buffer so that the memory module may serve requests continuously without having to wait to transmit reply packets back to their requesting processors. The sizes of the memory input and output buffers are assumed unbounded. A reply packet is assumed to be immediately consumed as soon as it arrives at the requesting processor.

Requests from different segments issued to a certain memory module can experience different average response times because of a non-uniform memory reference pattern or due to the non-symmetry of the network. However, requests generated by processors within the same segment will experience the same mean response times because the requests are indistinguishable. Hence, our proposed modeling approach employs a closed multi-class queueing network model to represent the SMBS. The number of packet classes is equal to the number of segments in the system. The service centers of the model represent the processors, memory modules and segment switches (together with their associated segment buses) of the SMBS. We make the following additional assumptions:

1. Each processor generates request packets with exponential distribution of inter-request interval with a mean of τ_p.

2. Read and write accesses in a memory module require the same service interval and take a constant time, τ_m.

3. A transient packet has priority over other packets for bus assignment. This assumption is based on the expectation that quickly delivered packets will reduce contention for network resources.

4. Arbitration time is included in a bus cycle time and a fair selection policy is utilized in each arbitration phase.

5. Bus service time is deterministic and is equal to a bus cycle time which is also equal to the flit time.

6. The queueing discipline at each memory queue is first-come first-served (FCFS). In the case when requests arrive simultaneously, these requests are served in random order.

As we discussed earlier, the SMBS supports wormhole routing. A segment switch begins forwarding a packet as soon as the header flit is received, provided that the flit buffer in the next segment switch can accept that flit and the next segment bus is available. Thus, the flits of a packet are transmitted from one segment switch to the next in a pipelined fashion and may occupy several segment switches and their associated segment buses along the path from source to destination. Only the header flit contains routing information. If the header flit is blocked because the flit

buffer in the next segment switch along the path is full or the associated segment bus is occupied by another packet, all the trailing flits of the blocked packet are blocked in place. Therefore, the segment switches and segment buses occupied by the flits are also blocked. If more than one flit can be buffered at a flit buffer, flits behind the header flit can catch-up at the flit buffer until the available buffer space is filled. At this point, they become blocked and can continue only after the header flit is unblocked. In the case that a header flit arrives at a target memory module, the remaining flits will always catch-up with it at the target memory module. We assume this method of routing throughout the dissertation.

When flit buffer capacity is unlimited, the wormhole routing scheme is equivalent to an optimized form of virtual cut-through switching. In virtual cut-through switching, if the header flit is blocked at a switch, the entire packet has to be buffered before the packet is forwarded. However, under the condition of unlimited flit buffer capacity, wormhole routing allows a partially received packet to be forwarded as soon as the header flit can advance.

Jackson [44] showed that when a queueing network consists of exponential queues, its solution is *separable*. In separable (or product form) queueing network models, each service center can be separated from the rest of the network and its solution could be evaluated in isolation. Then the solution of the entire network can be formed by combining these separate solutions. Our closed queueing network model, however, has features that violate separable model assumptions such as non-exponential queues, simultaneous possession of resources, etc.. Therfore, we will use

Table 3.1: Model input parameters.

Parameter	Description
g	Number of segments in an SMBS
n	Number of processors in a segment
m	Number of memory modules in a segment
b	Number of segment buses in a segment
L	Packet length
L_f	Length of a flit
t	Number of flits per packet
W_B	Segment bus bandwidth
τ_p	Mean time of request intervals
τ_m	Memory service time
P_{sd}	Probability that a request of class s is directed to a target memory module in segment d

Approximate Mean Value Analysis [16]. This technique has been shown to provide efficient and accurate solutions for non-separable models [16, 41, 45].

Model input parameters are summarized in Table 3.1. The SMBS consists of g identical segments numbered from 0 to $g-1$. Each segment is composed of n processors, m memory modules and b segment buses. The length of a packet is L. Each packet consists of t flits and each flit has length L_f. W_B denotes the segment bus bandwidth.

As we stated earlier, we assume that memory requests within a segment are uniformly distributed. Processors in the same segment are indistinguishable because they all experience the same mean response time. Therefore, all requests from processors in the same segment are of the same class. A request generated by a

processor in segment s (i.e., a class s request packet) directed to a memory module in segment d will have probability P_{sd}. The subscript sd in P_{sd} signifies a request of class s directed to a target memory module in segment d (in the forward path). Similarly, the subscript ds in P_{ds} will signify a reply to a class s request directed to the requesting processor in segment s from the target memory module in segment d (in the return path). If s and d are the same this would indicate an intra-segment access. If s is different from d, the request would correspond to an inter-segment access.

Sometimes, a segment switch in segment i will simply be called segment switch i. This convention can be applied to any system component like a processor, a segment bus or a memory module. An integer parameter which appears within square brackets, (for example, s in $R[s]$ or i in $R_{mem}[i]$) denotes the class of the request. The flit number will always appear within parentheses, as in $r_i(k)$.

3.2 Performance Issues

We shall study the performance of SMBS systems in detail under the model described in Section 3.1. We evaluate several parameters such as response time, processing efficiency, scalability of the SMBS under various configurations (various system sizes, flit buffer sizes and connection topologies) and under different workloads (varying memory request rates and uniform or non-uniform memory reference patterns). We begin by examining a *baseline* SMBS: a linear connection of segments. We shall study baseline systems for both the single flit buffer and the infinite flit buffer cases with a uniform memory reference workload. We shall also study baseline

systems with memory reference locality. Further we shall study the performance of an SMBS with a ring connection of segments with infinite flit buffers.

The flit buffer size is a design parameter that has cost and performance implications. The effect of flit buffer size on system cost is steadily diminishing because the price of memory is steadily decreasing. We shall compare system performance with single flit buffers against system performance with infinite flit buffers. These are the two extreme cases that establish bounds on the performance of systems with finite buffer sizes. The comparison will also help us to see how much system performance can be gained by increasing flit buffer size.

For interconnection networks with pipelined routing, like wormhole routing, modeling the performance with larger finite buffers is a difficult problem. This difficulty comes from the pipelining property of wormhole switching and the fact that blocking can take place due to the finite-sized flit buffers. In the SMBS with finite-sized flit buffers, for example, the remaining flits of a packet whose header flit is blocked simultaneously occupy several segment buses and switches. In addition, while the remaining flits are catching-up, the header flit may advance as soon as the next segment bus and flit buffer are available. All that makes it very difficult to develop an analytical model for such cases. Therefore, we study performance using simulation results only for the case of an SMBS with packet-sized flit buffers.

Our assumed model allows us to handle many types of memory request distributions. Figure 3.1 shows the relative segment bus traffic of an SMBS with 9 segments under the assumption that bus contention does not exist. Notice that segment bus

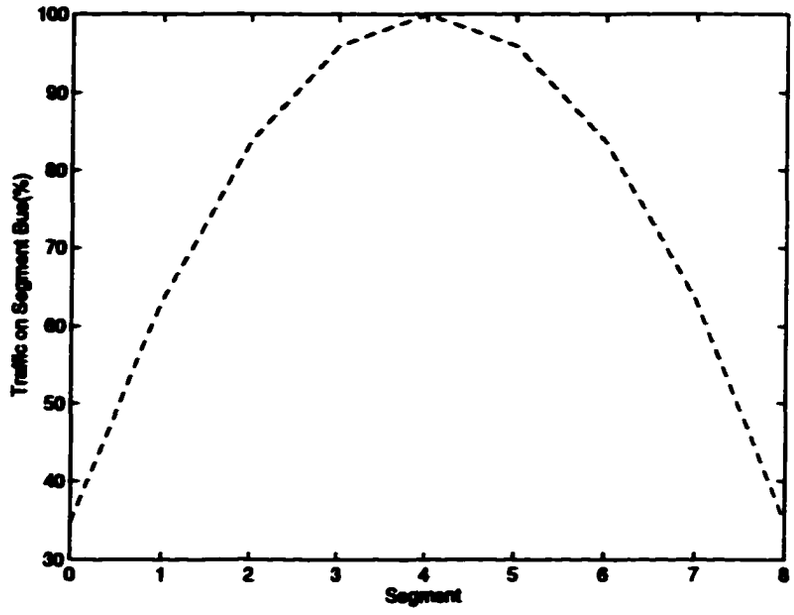


Figure 3.1: Relative segment bus traffic in an SMBS with 9 segments.

traffic is very unbalanced; the traffic is more congested in the middle than near the ends.

Applications exhibiting memory reference locality can take advantage of architectural features of the SMBS to realize performance gains. Applications with low locality place increasing bandwidth demands on the system. Increase in the bandwidth requirement in turn causes contention effects to become more profound. Hence, applications exhibiting reasonably high locality are likely to experience reduced network latency and contention. Figure 3.2 shows the relative segment bus traffic for an SMBS with 9 segments when memory reference locality is considered. Here, 75% locality means that probabilistically, 75% of requests are directed to local memory modules within a segment and the remaining 25% are evenly distributed to

non-local memory modules (modules outside the segment). As can be clearly seen, bus traffic balance is improved considerably as locality rate increases.

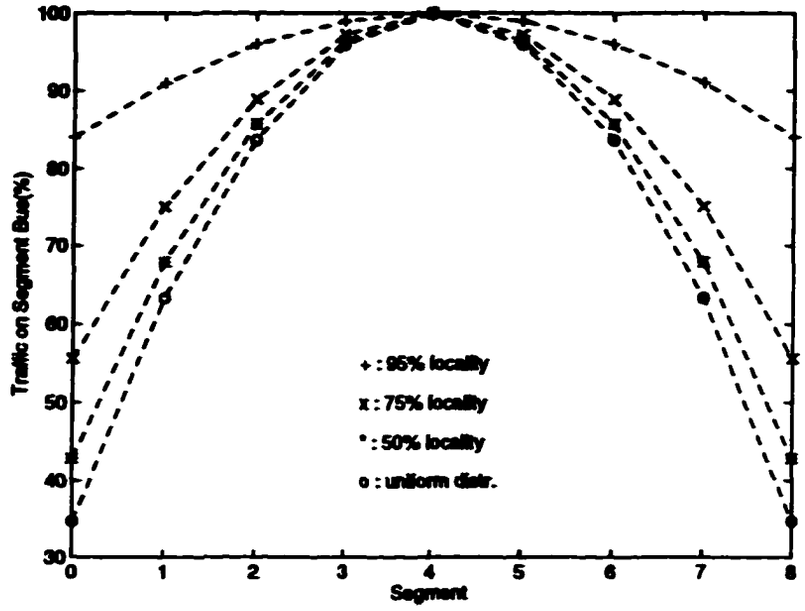


Figure 3.2: Relative segment bus traffic in an SMBS with 9 segments with varying locality.

3.3 Overview of the Model

The mean response time of a request issued to a target memory module will be used as a performance measure. Processors belonging to different classes may experience distinct response times because of the non-symmetry of the network as well as, possibly, non-uniform memory reference patterns.

Consider a request of class s for a memory module located in segment d. A processor will generate a request packet of class s traveling in its forward path from s to d and will receive a reply packet of the same class from the targeted memory module in segment d. The mean residence time experienced by the request in its

forward trip to the target memory and that experienced by the reply in the return trip is dependent on the request's class. Hence, the mean response time of a request from a processor in segment s (a class s request) is

$$R[s] = R_{proc}[s] + R_{network}[s] + R_{mem}[s] \qquad s = 0, \ldots, g-1. \tag{3.1}$$

$R[s]$ is the sum of the mean residence times (queueing and service) for a class s request packet in the processor $R_{proc}[s]$, in the network $R_{network}[s]$ and in the memory module $R_{mem}[s]$. In short, we call $R[s]$ the mean response time of a class s request. The overall mean response time is given by

$$R = \frac{1}{g}\sum_{s=0}^{g-1} R[s]. \tag{3.2}$$

$R_{proc}[s]$ denotes the mean residence time that the header flit of a class s packet experiences at the head of a processor output queue. This includes the mean waiting time for a segment bus plus the time for transferring the header flit to the first segment switch (in the case of an inter-segment access) or the time for transferring the header flit on a segment bus to a local target memory module (in the case of an intra-segment access).

The residence time in the network is the weighted sum of the mean residence times in the forward path (for a request packet) and in the return path (for a reply).

Thus,

$$R_{network}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{sd,network} + R_{ds,network}) \qquad s = 0, \ldots, g-1, \qquad (3.3)$$

where $\sum_{d=0}^{g-1} P_{sd} = 1$. $R_{sd,network}$ denotes the average residence time in the network of a class s request packet from the first segment switch on the forward path to the target memory in segment d. $R_{sd,network}$ denotes the average residence time in the network of a class s reply packet from the first segment switch in the return path to the processor that originated the request.

The residence time at a segment switch consists of the mean delay spent in the flit buffer waiting for a segment bus and the time for transferring the header flit to the next switch (for intermediate segment switches) or to the target memory module (for the final segment switch in the forward path) or to the original requesting processor (for the final segment switch in the return path).

The residence time in the network includes the catch-up times which occur both in the forward and return paths. The catch-up time in the forward path is defined as the mean delay until all the remaining flits reach the destination after the header flit arrives at the target memory module. The catch-up time in the return path is defined as the mean delay until all the remaining flits reach the processor after the header flit returns to the requesting processor. In the case of an intra-segment access, the residence time in the network is the sum of the catch-up times in the forward and return paths because a packet does not traverse any segment switches.

The residence time in a memory module is given by

$$R_{mem}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{d|s,mem_{in}} + R_{d|s,mem_{out}}) \qquad s = 0, \ldots, g-1. \tag{3.4}$$

$R_{d|s,mem_{in}}$ is the sum of the average delay that a class s request packet experiences in the input queue of the target memory module (in segment d) and the memory service time, τ_m. $R_{d|s,mem_{out}}$ consists of two parts. For an inter-segment access, it includes the mean queueing delay of a class s reply packet in the memory output queue, the mean waiting time for a segment bus at the head of the memory output queue and the time for transferring the header flit to the first segment switch in the return path. For an intra-segment access, it consists of the mean queueing delay at the memory output queue, the segment bus waiting time at the head of the memory output queue and the time for transferring the reply header flit back to its local processor.

We will also utilize another performance metric known as *processing efficiency* to evaluate SMBS's. Processing efficiency is defined as the fraction of time a processor is busy. If a processor generates request messages with the mean interval, τ_p, then the mean processor utilization of class s, $\rho_{proc}[s]$, can be expressed as

$$\rho_{proc}[s] = \frac{\tau_p}{\tau_p + R[s]} \qquad s = 0, \ldots, g-1, \tag{3.5}$$

where $R[s]$ is the mean response time of a class s request. Notice that the above expression does not distinguish among processors in the same class (segment) which is consistent with our model.

3.4 Residence Time in the Network

In this section, we evaluate network residence time for the SMBS. In the case of an inter-segment access, $R_{sd,network}$ and $R_{ds,network}$ correspond to the average residence times for a packet of class s in the network in the forward and return paths, respectively. The mean residence time for a class s packet in the forward path consists of the following two elements:

- The sum of the mean residence times of the header flit of a class s packet at all segment switches in the forward path from s to d, $\sum_i r_{i,sd}(1)$, where $r_{i,sd}(1)$ refers to the mean residence time of the header flit of a class s packet at segment switch i in the forward path from s to d.

- The catch-up time, defined as the mean delay until all the remaining flits reach the destination after the header flit has arrived at the addressed target memory module in segment d ($\frac{L-L_f}{W_B}$).

Therefore, the mean residence time in the forward path can be written as

$$R_{sd,network} = \sum_i r_{i,sd}(1) + \frac{L - L_f}{W_B}.$$

The limit of the summation index in the first term is dependent on the fact that a request may take a downstream path ($d - s < 0$) or an upstream path ($d - s > 0$).

For a forward path, the mean network residence time is given by

$$R_{sd,network} = \begin{cases} \sum_{i=s+1}^{d} r_{i,sd}(1) + \frac{L - L_f}{W_B} & \text{if } d - s > 0, \\ \sum_{i=d+1}^{s} r_{i,sd}(1) + \frac{L - L_f}{W_B} & \text{if } d - s < 0. \end{cases} \tag{3.6}$$

For a return path, the mean network residence time is

$$R_{ds,network} = \begin{cases} \sum_{i=s+1}^{d} r_{i,ds}(1) + \frac{L - L_f}{W_B} & \text{if } d - s > 0, \\ \sum_{i=d+1}^{s} r_{i,ds}(1) + \frac{L - L_f}{W_B} & \text{if } d - s < 0. \end{cases} \tag{3.7}$$

In the case of an intra-segment access ($s = d$), $r_{i,sd}(1)$ and $r_{i,ds}(1)$ are equal to zero.

3.4.1 Mean Flit Residence Time at a Segment Switch

Suppose that the k^{th} flit, $k > 1$, resides at segment switch i. Recall that a segment switch in segment i will simply be called segment switch i. A segment switch on the path to a given destination which is traversed last by the header flit will be called *the last segment switch*. If the last segment switch is $k-1$ or more hops away from the current segment switch i, the mean residence time of the k^{th} flit at the latter segment switch can be estimated as the mean residence time of the header flit at segment switch $i + k - 1$ in the case of an upstream forward path or that at segment switch $i - k + 1$ in the case of a downstream forward path. This method for calculating the k^{th} flit's residence time by estimating the header flit's residence

time is similar to that used in [41]. This is clearly due to the pipelining property of the wormhole routing technique. Here, the number of hops is defined as the number of segment buses that a header flit must traverse to reach the destination. If the header flit has already reached the addressed target memory module, the residence time of the k^{th} flit at a segment switch is the same as a single flit transfer time on a segment bus.

Let $r_{i,sd}(k)$ denote the mean residence time of the k^{th} flit of a class s request packet at a segment switch in segment i in the forward path from s to d. Then $r_{i,sd}(k)$ is given by

$$r_{i,sd}(k) = \begin{cases} r_{i+k-1,sd}(1) & \text{if } d \text{ is } k-1 \text{ or more hops upstream away from } i, \\ r_{i-k+1,sd}(1) & \text{if } d \text{ is } k \text{ or more hops downstream away from } i, \\ \frac{L_f}{W_B} & \text{otherwise.} \end{cases} \quad (3.8)$$

Similarly, let $r_{i,ds}(k)$ denote the mean residence time of the k^{th} flit of a class s reply packet at a segment switch in segment i in the return path from d to s. $r_{i,ds}(k)$ can be expressed as

$$r_{i,ds}(k) = \begin{cases} r_{i+k-1,ds}(1) & \text{if } s \text{ is } k-1 \text{ or more hops upstream away from } i, \\ r_{i-k+1,ds}(1) & \text{if } s \text{ is } k \text{ or more hops downstream away from } i, \\ \frac{L_f}{W_B} & \text{otherwise.} \end{cases} \quad (3.9)$$

The mean residence time of the header flit of a class s packet at segment switch i in the forward path from s to d, $r_{i,sd}(1)$, consists of the following three components:

1. **The average waiting time for a segment bus at segment i ($w_i(1)$).**

2. **The mean residual residence time of the tail flit (of a preceding packet) at the next switch along the path.**

3. **The time for transferring the header flit to the next segment switch: a flit time ($\frac{L_f}{W_B}$).**

Note that when the header flit arrives at the last segment switch (either segment d on an upstream forward path or segment $d+1$ on a downstream forward path), $r_{i,sd}(1)$ will not include the mean residual residence time of a tail flit because the flit finds an unbounded memory input queue. Similarly, on a return path, $r_{i,ds}(1)$ will not include the mean residual residence time of a tail flit when i equals s for upstream flow or when i equals $s+1$ for downstream flow.

The calculations of those terms are dependent on whether the header flit takes an upstream or a downstream path. Depending on $\mathcal{D}$ $(= d - s)$, the path will consist of segment buses either from the upper bus group (if $\mathcal{D} > 0$) or from the lower bus group (if $\mathcal{D} < 0$). Hence, $r_{i,sd}(1)$ can have two different values, depending on whether the header flit takes an upstream path or a downstream path. We shall use $r_{i,sd}(1)$ without using an explicit indication as to whether the path is an upstream or a downstream path. When an explicit distinction is necessary, the superscripts $+$ and $-$ will be used to denote upstream and downstream flow, respectively.

Let $r_i(k)$ denote the average residence time of the k^{th} flit at a segment switch in segment i. $u_i(k)$ is defined as the k^{th} flit's mean utilization of the segment switch. Upstream flit buffers are utilized when a packet takes an upstream forward path or

an upstream return path. Downstream flit buffers are utilized when a packet moves downstream (whether it is a forward or a return path). At upstream flit buffer i, $r_i(k)$ and $u_i(k)$ are expressed as

$$r_i(k) = \frac{\sum_{s=0}^{g-1} \sum_{d \in D^+_{i,sd}} P_{sd} r_{i,sd}(k) + \sum_{s=0}^{g-1} \sum_{d \in D^-_{i,sd}} P_{sd} r_{i,ds}(k)}{\sum_{s=0}^{g-1} \sum_{d \in D^+_{i,sd}} P_{sd} + \sum_{s=0}^{g-1} \sum_{d \in D^-_{i,sd}} P_{sd}}, \tag{3.10}$$

$$u_i(k) = \sum_{s=0}^{g-1} \sum_{d \in D^+_{i,sd}} \frac{n P_{sd} r_{i,sd}(k)}{\tau_p + R[s]} + \sum_{s=0}^{g-1} \sum_{d \in D^-_{i,sd}} \frac{n P_{sd} r_{i,ds}(k)}{\tau_p + R[s]}. \tag{3.11}$$

At downstream flit buffer i, $r_i(k)$ and $u_i(k)$ are expressed as

$$r_i(k) = \frac{\sum_{s=0}^{g-1} \sum_{d \in D^-_{i,sd}} P_{sd} r_{i,sd}(k) + \sum_{s=0}^{g-1} \sum_{d \in D^+_{i,sd}} P_{sd} r_{i,ds}(k)}{\sum_{s=0}^{g-1} \sum_{d \in D^-_{i,sd}} P_{sd} + \sum_{s=0}^{g-1} \sum_{d \in D^+_{i,sd}} P_{sd}}, \tag{3.12}$$

$$u_i(k) = \sum_{s=0}^{g-1} \sum_{d \in D^-_{i,sd}} \frac{n P_{sd} r_{i,sd}(k)}{\tau_p + R[s]} + \sum_{s=0}^{g-1} \sum_{d \in D^+_{i,sd}} \frac{n P_{sd} r_{i,ds}(k)}{\tau_p + R[s]}, \tag{3.13}$$

where

$$D^+_{i,sd} \equiv \{d \mid \text{packets traveling from } s \text{ to } d \text{ will visit upstream flit buffer } i\}$$

$$D^-_{i,sd} \equiv \{d \mid \text{packets traveling from } s \text{ to } d \text{ will visit downstream flit buffer } i\}.$$

Next, we discuss the derivation of each term in the flit residence time. For this discussion we assume that a packet is on its upstream forward path. We call a

header flit waiting for a segment bus at segment switch i the *tagged header flit*, in order to distinguish it from others. The tagged header flit must wait for a segment bus to be available if all buses are occupied by other packets. The waiting time for a segment bus, $w_i(1)$, will be discussed later.

Once the tagged header flit acquires a segment bus, it may observe the tail flit of another packet in service at the next flit buffer (which is located at segment switch $i+1$). The residual residence time of a tail flit is a random variable whose distribution depends on the distribution of the original residence time. To estimate the residual residence time, we assume that the tagged header flit is equally likely to arrive at a flit buffer at any point during the residence time interval of a tail flit in the next flit buffer along the path. Under this assumption, the residual residence time is given by

$$\frac{r_{i+1}(t)}{2} + \frac{\sigma^2}{2r_{i+1}(t)}$$

where σ^2 is the variance in the average residence time of a tail flit at segment switch $i+1$.

A reasonable determination of the residual residence time is not easy because the first and second moments of the residence time distribution (which is unknown) must be calculated. Hence, we assume that the residence time of each flit at flit buffer i is *deterministic* and that its value is given by $r_i(k)$. Under this assumption, the residual residence time would be equal to one half of the residence time, i.e., $r_{i+1}(t)/2$. The error introduced by this assumption would be small and could be

ignored because the mean residual residence time is a small component in the mean residence time of the tagged header flit.

The term for the mean residual residence time relates directly to the pipelining property of wormhole routing. The tagged flit may only observe the tail flit of a preceding packet in the flit buffer of the next segment switch, if it observes any. Hence, the mean residual residence time can be calculated by multiplying the mean utilization of the flit buffer by a tail flit, $u_{i+1}(t)$, and the mean residual residence time of a tail flit, $r_{i+1}(t)/2$. Note that when a segment switch in segment i becomes the last segment switch on the path to the destination ($i = d$ on an upstream path and $i = d+1$ on a downstream path), $r_{i,sd}(1)$ will not include the residual residence time of the tail flit because the flit finds an unbounded memory input queue. Therefore, the residence time, $r_{i,sd}(1)$ on an upstream forward path is given by

$$r_{i,sd}(1) = \begin{cases} w_i(1) + u_{i+1}(t)\dfrac{r_{i+1}(t)}{2} + \dfrac{L_f}{W_B} & \text{for } i \neq d, \\ \\ w_i(1) + \dfrac{L_f}{W_B} & \text{for } i = d. \end{cases} \tag{3.14}$$

The residence time, $r_{i,ds}(1)$ on a downstream return path is given by

$$r_{i,ds}(1) = \begin{cases} w_i(1) + u_{i-1}(t)\dfrac{r_{i-1}(t)}{2} + \dfrac{L_f}{W_B} & \text{for } i \neq s+1, \\ \\ w_i(1) + \dfrac{L_f}{W_B} & \text{for } i = s+1. \end{cases} \tag{3.15}$$

Similarly, the residence time, $r_{i,sd}(1)$ on a downstream forward path is given by

$$r_{i,sd}(1) = \begin{cases} w_i(1) + u_{i-1}(t)\frac{r_{i-1}(t)}{2} + \frac{L_f}{W_B} & \text{for } i \neq d+1, \\ \\ w_i(1) + \frac{L_f}{W_B} & \text{for } i = d+1, \end{cases} \tag{3.16}$$

and $r_{i,ds}(1)$ on an upstream return path is given by

$$r_{i,ds}(1) = \begin{cases} w_i(1) + u_{i+1}(t)\frac{r_{i+1}(t)}{2} + \frac{L_f}{W_B} & \text{for } i \neq s, \\ \\ w_i(1) + \frac{L_f}{W_B} & \text{for } i = s. \end{cases} \tag{3.17}$$

3.4.2 Average Waiting Time for a Segment Bus

In computing the average waiting time for a segment bus, we must examine the arbitration scheme. We still maintain the assumption that the tagged header flit is on its upstream forward path; that is, the flit is in an upstream flit buffer of a segment switch connected to a bus in the upper bus group. That implies that the flit will be involved in the upper bus group arbitration. The tagged flit at a segment switch is always involved in the first phase of arbitration. According to our model, if a segment bus is not occupied by another packet, the concerned segment bus will always be available to the tagged flit. Hence, waiting for a segment bus is experienced by the tagged flit only when the segment bus is occupied. For such a case, the tagged flit must wait until the segment bus is released.

Let $u_{i,p_{out}}(k)$ denote the mean utilization of a processor output queue by the k^{th} flit of class i. Also let $u_{i,m_{out}}(k)$ denote the mean utilization of a memory output queue in segment i by the k^{th} flit.

Now, let us try to find the average waiting time. Note that the waiting time will be different for different segments. We begin by examining the last segment. Suppose that the tagged header flit is blocked at the segment switch of segment $g-1$. This would occur when a local processor or a local memory module occupies the segment bus. For this case, the average waiting time is given by

$$w_{g-1}(1) = \sum_{k=2}^{t} \left(\frac{u_{g-1,p_{out}}(k)}{2} + \frac{u_{g-1,m_{out}}(k)}{2} \right) \frac{P_{g-1,g-1}}{2} \left(\frac{L_f}{2W_B} + (t-k)\frac{L_f}{W_B} \right).$$

The two terms in the first pair of parentheses represent the probability that when the tagged header flit arrives at a segment switch, it observes its associated segment bus occupied by the k^{th} flit of a memory request packet from a processor in segment $g-1$ or the k^{th} flit of a reply packet from a memory module in segment $g-1$, respectively. The reason why the summation index k begins with "$k=2$" is that a transient packet at a segment switch has priority over other packets (i.e., the packet can only observe an already occupied segment bus).

Let the flit observed by the header flit when it arrived at a segment switch in segment $g-1$ be the k^{th} flit. Then the terms in the second pair of parentheses correspond to the mean residual segment bus access time observed by the tagged flit including the k^{th} flit it observed on arrival. At segment $g-1$, the mean residual time is contributed to by one half the probability of an intra-segment access: $\frac{P_{g-1,g-1}}{2}$.

The average waiting time for a segment bus at segment $g-2$ is expressed as

$$w_{g-2}(1) =$$

$$\left(\frac{u_{g-2,p_{out}}(2)}{2}+\frac{u_{g-2,m_{out}}(2)}{2}\right)\left\{\frac{P_{g-2,g-2}}{2}\left(\frac{L_f}{2W_B}+(t-2)\frac{L_f}{W_B}\right)\right.$$

$$\left.+\ P_{g-2,g-1}\left(\frac{r_{g-1,(g-2)(g-1)}(1)}{2}+(t-2)\frac{L_f}{W_B}\right)\right\}$$

$$+\sum_{k=3}^{t}\left(\frac{u_{g-2,p_{out}}(k)}{2}+\frac{u_{g-2,m_{out}}(k)}{2}\right)\left(\frac{P_{g-2,g-2}}{2}+P_{g-2,g-1}\right)\left(\frac{L_f}{2W_B}+(t-k)\frac{L_f}{W_B}\right).$$

Suppose that the tagged header flit observes the second flit of a request or a reply packet occupying the segment bus of segment $g-2$. In the case of an intra-segment access, the mean residual bus access time of the remaining flits, including the second flit is, $\frac{L_f}{2W_B}+(t-2)\frac{L_f}{W_B}$. In the case of an inter-segment access, the tagged header flit will, on average, wait for a period equal to the average residence time of the header flit at segment switch $g-1$ plus the residual bus access time of the remaining flits. The last term in the above expression represents the mean time elapsed until the segment bus is released by the remaining flits assuming that the incoming tagged flit observed the k^{th} flit ($k \geq 3$) occupying the segment bus.

Before we generalize, we go one step further and express the waiting time at segment $g-3$ as

$$w_{g-3}(1) =$$

$$\left(\frac{u_{g-3,p_{out}}(2)}{2} + \frac{u_{g-3,m_{out}}(2)}{2}\right)\left\{\frac{P_{g-3,g-3}}{2}\left(\frac{L_f}{2W_B} + (t-2)\frac{L_f}{W_B}\right)\right.$$

$$+ P_{g-3,g-2}\left(\frac{r_{g-2,(g-3)(g-2)}(1)}{2} + (t-2)\frac{L_f}{W_B}\right)$$

$$\left. + P_{g-3,g-1}\left(\frac{r_{g-2,(g-3)(g-1)}(1)}{2} + (t-2)\frac{L_f}{W_B}\right)\right\}$$

$$+ \left(\frac{u_{g-3,p_{out}}(3)}{2} + \frac{u_{g-3,m_{out}}(3)}{2}\right)\left\{\left(\frac{P_{g-3,g-3}}{2} + P_{g-3,g-2}\right)\left(\frac{L_f}{2W_B} + (t-3)\frac{L_f}{W_B}\right)\right.$$

$$\left. + P_{g-3,g-1}\left(\frac{r_{g-1,(g-3)(g-1)}(1)}{2} + (t-3)\frac{L_f}{W_B}\right)\right\}$$

$$+ \sum_{k=4}^{t}\left(\frac{u_{g-3,p_{out}}(k)}{2} + \frac{u_{g-3,m_{out}}(k)}{2}\right)\left(\frac{P_{g-3,g-3}}{2} + \sum_{j=g-2}^{g-1} P_{g-3,j}\right)\left(\frac{L_f}{2W_B} + (t-k)\frac{L_f}{W_B}\right).$$

Generalization of the expression for the average waiting time for a segment bus of the upper bus group at segment i is thus given by

$$w_i(1) =$$

$$\sum_{k=2}^{min(t,g-i)}\left(\frac{u_{i,p_{out}}(k)}{2} + \frac{u_{i,m_{out}}(k)}{2}\right)\left\{\frac{P_{ii}}{2}\left(\frac{L_f}{2W_B} + (t-k)\frac{L_f}{W_B}\right)\right.$$

$$+ \sum_{j=i+1}^{g-1} P_{ij} \left(\frac{r_{i+k-1,ij}(1)}{2} + (t-k)\frac{L_f}{W_B} \right) \Bigg\}$$

$$+ \sum_{k=g-i+1}^{t} \left(\frac{u_{i,p_{out}}(k)}{2} + \frac{u_{i,m_{out}}(k)}{2} \right) \left(\sum_{j=i+1}^{g-1} P_{ij} + \frac{P_{ii}}{2} \right) \left(\frac{L_f}{2W_B} + (t-k)\frac{L_f}{W_B} \right). \tag{3.18}$$

Similarly, the expression for the average waiting time for segment bus i of a lower bus group is given as

$w_i(1) =$

$$\sum_{k=2}^{min(t,i)} \left(\frac{u_{i-1,p_{out}}(k)}{2} + \frac{u_{i-1,m_{out}}(k)}{2} \right) \left\{ \frac{P_{i-1,i-1}}{2} \left(\frac{L_f}{2W_B} + (t-k)\frac{L_f}{W_B} \right) \right.$$

$$\left. + \sum_{j=0}^{i-2} P_{i-1,j} \left(\frac{r_{i+1-k,(i-1)j}(1)}{2} + (t-k)\frac{L_f}{W_B} \right) \right\}$$

$$+ \sum_{k=i+1}^{t} \left(\frac{u_{i-1,p_{out}}(k)}{2} + \frac{u_{i-1,m_{out}}(k)}{2} \right) \left(\sum_{j=0}^{i-2} P_{i-1,j} + \frac{P_{i-1,i-1}}{2} \right) \left(\frac{L_f}{2W_B} + (t-k)\frac{L_f}{W_B} \right). \tag{3.19}$$

3.5 Residence Time in a Processor

Suppose that a memory request at the head of a processor output queue is looking for a segment bus of an upper bus group in segment i. This request could be an intra-segment or an inter-segment access request and must participate in the second phase of the arbitration.

In the case of an intra-segment access, the processor must undergo memory arbitration as well as bus arbitration. A request from processor i participates with probability $p_{i,proc|mem} = u_{i,p_{out}}(1)P_{ii}/m$ in memory arbitration. Up to n such requests could participate in memory arbitration. To simplify the notation, we will drop the subscript i denoting segment i in the expressions to follow. If a segment switch at segment i contains a flit directed to a memory module in segment i, then this switch must also participate in memory arbitration. The probability that such a segment switch will have to participate in memory arbitration is given by

$$p_{sw_{up}|mem} = \frac{u_i(1)\sum_{s=0}^{i-1}\frac{P_{si}}{m}}{\sum_{s=0}^{i-1}\sum_{d=i}^{g-1}P_{sd}}. \tag{3.20}$$

The above probability represents the fraction of segment switch utilization contributed by terminating packets (packets terminating at the segment under consideration). In a lower bus group, the probability that a segment switch will participate in memory arbitration is given by

$$p_{sw_{lo}|mem} = \frac{u_{i+1}(1)\sum_{s=i+1}^{g-1}\frac{P_{si}}{m}}{\sum_{s=i+1}^{g-1}\sum_{d=0}^{i}P_{sd}}. \tag{3.21}$$

We will assume that the requests from processors and segment switches are independent Bernoulli trials. Therefore, we can consider a distribution which deals with the number of requests issued to local memory modules by processors or segment switches, separately. Let μ_i denote the number of requests issued to local memory

module i $(1 \leq i \leq m)$ by processors. We will conveniently denote the state of memory requests by processors using an m-tuple vector $\vec{\mu}_p = (\mu_1, \ldots, \mu_m)$ in which $0 \leq \mu_i \leq n$ for $1 \leq i \leq m$ and $0 \leq \mu = \sum_{1 \leq i \leq m} \mu_i \leq n$. Each μ_i represents the number of processor requests issued to memory module i in the segment under consideration. Thus, the set of all feasible states corresponding to the number of requests issued to each of m local memory modules by n processors in a given segment is defined as

$$S_{\vec{\mu}_p}(m,n) = \{(\mu_1, \ldots, \mu_m) \mid 0 \leq \mu_i \leq n \text{ for } 1 \leq i \leq m \text{ and } 0 \leq \mu = \sum_{1 \leq i \leq m} \mu_i \leq n\}. \tag{3.22}$$

Notice that μ corresponds to the total number of requests issued by processors to the m memory modules in the segment under consideration. As we mentioned earlier, processor requests are modeled as Bernoulli trials. Thus, the requests follow a multinomial distribution, denoted by $P_{\vec{\mu}_p}(n)$ and can be expressed as

$$P_{\vec{\mu}_p}(n) = \frac{n! p_{proc|mem}^{\mu} (1 - m p_{proc|mem})^{n-\mu}}{(n-\mu)! \prod_{1 \leq i \leq m} \mu_i!}. \tag{3.23}$$

Let ν_i and ξ_i denote the number of requests issued to local memory module i $(1 \leq i \leq m)$ by the segment switches of the upper and lower bus groups, respectively. The states representing the number of requests issued to each of the m local memory modules by the $b/2$ upper segment switches and $b/2$ lower segment switches will be

represented as m-tuple vectors $\vec{\nu}_s = (\nu_1, \ldots, \nu_m)$ and $\vec{\xi}_s = (\xi_1, \ldots, \xi_m)$, respectively, where each ν_i (ξ_i) corresponds to the number of requests to memory module i from the upper (lower) bus group. The set of all feasible states corresponding to the number of requests issued to each of m local memory modules by the $b/2$ upper segment switches is thus

$$S_{\vec{\nu}_s}(m, \frac{b}{2}) = \{(\nu_1, \ldots, \nu_m) \mid 0 \leq \nu_i \leq \frac{b}{2} \text{ for } 1 \leq i \leq m \text{ and } 0 \leq \nu = \sum_{1 \leq i \leq m} \nu_i \leq \frac{b}{2}\}. \tag{3.24}$$

Similarly, the set of all feasible states corresponding to the number of requests issued to each of m local memory modules by the $b/2$ lower segment switches is given by

$$S_{\vec{\xi}_s}(m, \frac{b}{2}) = \{(\xi_1, \ldots, \xi_m) \mid 0 \leq \xi_i \leq \frac{b}{2} \text{ for } 1 \leq i \leq m \text{ and } 0 \leq \xi = \sum_{1 \leq i \leq m} \xi_i \leq \frac{b}{2}\}. \tag{3.25}$$

As with processor requests, the requests by the $b/2$ upper segment switches and $b/2$ lower segment switches will also follow multinomial distributions, denoted by $p_{\vec{\nu}_s}(\frac{b}{2})$ and $p_{\vec{\xi}_s}(\frac{b}{2})$, respectively, and can be expressed as follows.

$$p_{\vec{\nu}_s}(\frac{b}{2}) = \frac{\frac{b}{2}! p_{sw_{up}|mem}^{\nu} (1 - m p_{sw_{up}|mem})^{\frac{b}{2}-\nu}}{(\frac{b}{2} - \nu)! \prod_{1 \leq i \leq m} \nu_i!} \tag{3.26}$$

$$p_{\vec{\xi}_s}(\frac{b}{2}) = \frac{\frac{b}{2}! p_{sw_{lo}|mem}^{\xi} (1 - m p_{sw_{lo}|mem})^{\frac{b}{2}-\xi}}{(\frac{b}{2}-\xi)! \prod_{1 \le i \le m} \xi_i!} \tag{3.27}$$

Without loss of generality, we can assume that the m^{th} memory module is the tagged memory module because the m memory modules in a segment are indistinguishable from one another for the purpose of our analysis. Let $\vec{K}_{tag}$ denote a vector representing the number of requests issued to the tagged memory module by processors and segment switches using the upper and lower bus groups. A feasible state for module m can be described by a ternary vector $\vec{K}_{tag} = (\mu_m, \nu_m, \xi_m)$ in which $0 \le \mu_m \le n$, $0 \le \nu_m \le \frac{b}{2}$, $0 \le \xi_m \le \frac{b}{2}$ and $0 \le K_{tag} = \mu_m + \nu_m + \xi_m \le n + b$. Clearly K_{tag} corresponds to the total number of requests from any source to module m. The set of all feasible states which represent the number of requests issued to the tagged memory module by n processors, $b/2$ upper segment switches and $b/2$ lower segment switches in a segment can thus be defined as

$$S_{\vec{\mu}_p \vec{\nu}_s \vec{\xi}_s}(n, \frac{b}{2}, \frac{b}{2}) = \{(\mu_m, \nu_m, \xi_m) \mid 0 \le \mu_m \le n, 0 \le \nu_m, \xi_m \le \frac{b}{2} \text{ and } 0 \le K_{tag} \le n + b\} \tag{3.28}$$

where $K_{tag} = \mu_m + \nu_m + \xi_m$.

To the tagged memory module, a total of K_{tag} memory requests is issued. Therefore, the probability that a request from a processor succeeds in memory arbitration

given that up to n processors and b segment switches might compete is given by

$$p_{p_{succ}|mem} = \sum_{\vec{K}_{tag} \in S_{\vec{u}_p \vec{v}_s \vec{\xi}_s}} \frac{1}{1+K_{tag}} p_{\vec{u}_p}(n-1) p_{\vec{v}_s}(\frac{b}{2}) p_{\vec{\xi}_s}(\frac{b}{2}). \tag{3.29}$$

Now let us find the cases which require participation in the upper bus group arbitration of segment i. Each inter-segment request by a processor will participate in bus arbitration with probability

$$p_{proc|bus} = u_{i,p_{out}}(1) \sum_{j=i+1}^{g-1} P_{ij}. \tag{3.30}$$

Similarly, a reply packet from a memory module takes part in bus arbitration with probability

$$p_{mem|bus} = u_{i,m_{out}}(1) \left(\frac{P_{ii}}{2} + \sum_{j=i+1}^{g-1} P_{ij} \right). \tag{3.31}$$

In the above equation, the probability $P_{ii}/2$ represents the proportion of cases in which a reply packet (of an intra-segment access) participates either in the upper bus group or in the lower bus group arbitration.

As discussed before, in the case of an intra-segment access, an outstanding request from a processor which succeeds in memory arbitration must take part in bus arbitration to eventually get access to a segment bus. Therefore, the effective

probability that a processor successfully participates in bus arbitration is given by

$$p_{proc|bus_{eff}} = \frac{P_{ii}}{2} u_{i,p_{out}(1)} p_{succ|mem} + p_{proc|bus}. \quad (3.32)$$

The first term gives the probability that an intra-segment memory request will succeed in memory arbitration and will participate in bus arbitration. The second term represents the probability that an inter-segment request participates in bus arbitration.

The delay experienced in a processor output queue by a class s header flit, basically, consists of the following two components:

1. The mean waiting time of the header flit of a class s packet directed to d in order to compete for a segment bus at the processor output queue in segment s ($s_{d|s,p_{out}}$).

2. The flit time for transferring the header flit to the first segment switch on the path to d, for an inter-segment access, or the time for transferring the header flit to its addressed local memory module, for an intra-segment access ($\frac{L_f}{W_B}$).

In the case of an inter-segment request, the residence time must include the mean residual residence time of a tail flit in service because the header flit may observe the tail flit of a preceding packet when it arrives at the first segment switch in the path from s to d ($\gamma_{d|s,p_{out}}$). Hence, the mean residence time for the header flit of

class s at the processor output queue is given by

$$R_{proc}[s] = \sum_{d=0}^{g-1} P_{sd} s_{d|s,p_{out}} + \sum_{\substack{d=0 \\ d \neq s}}^{g-1} P_{sd} \gamma_{d|s,p_{out}} + \frac{L_f}{W_B} \qquad s = 0, \ldots, g-1. \tag{3.33}$$

The mean residual residence time, $\gamma_{d|s,p_{out}}$, is calculated in the same way as in the previous section and can be expressed as

$$\gamma_{d|s,p_{out}} = \begin{cases} u_{s+1}(t)\dfrac{r_{s+1}(t)}{2} & \text{if } d-s>0, \\ \\ u_s(t)\dfrac{r_s(t)}{2} & \text{if } d-s<0. \end{cases} \tag{3.34}$$

Consider a class s request. When such a request arrives at the head of a processor output queue, it will observe $\mathcal{K}_{s,p_{out}}$ other requests occupying the heads of their respective queues, where $\mathcal{K}_{s,p_{out}}$ is an integer random variable. The occupied queue heads are from a mix of processor output queues and memory output queues. Here, the term "occupied queue head" implies a state in which a request that already arrived at the head of an output queue waits for segment bus assignment. If $\mathcal{K}_{s,p_{out}}$ is smaller than the number of available free segment buses in a bus group, the request doesn't need to wait for a segment bus assignment. However, if $\mathcal{K}_{s,p_{out}}$ is equal to or greater than the number of free segment buses, the request should wait until a free segment bus is available. Suppose that i is the number of free buses observed by the header flit when it arrives at the head of a processor output queue. If $\mathcal{K}_{s,p_{out}}$ is greater than i, the request should wait $(\mathcal{K}_{s,p_{out}} - i)$ service completions before it can get access to a segment bus. This is because when a bus becomes free, the

next request to be granted the bus is selected according to a FCFS policy. Cyclic selection is used for simultaneous arrivals.

The mean segment bus access time is the time for transferring a request or a reply packet on a segment bus. For an intra-segment request, the segment bus access time will be L/W_B. For an inter-segment request, the bus access time is the sum of the transfer time of a header flit on a segment bus and the mean residence time of the k^{th} flit for $1 \leq k < t$ at the first segment switch. This is equal to the time elapsed until a segment bus is released. The mean segment bus access time at segment s will be denoted by $t_{s,bus_{up}}$ or $t_{s,bus_{lo}}$, where the subscript bus_{up} implies a segment bus in the upper bus group and bus_{lo} implies a segment bus in the lower bus group. These times are given by

$$t_{s,bus_{up}} = \frac{\frac{nP_{ss}}{2m}\frac{L}{W_B} + \sum_{d=s+1}^{g-1} nP_{sd}\left\{\frac{L_f}{W_B} + \sum_{k=1}^{t-1} r_{s+1,sd}(k)\right\}}{\frac{nP_{ss}}{2m} + \sum_{d=s+1}^{g-1} nP_{sd}}, \tag{3.35}$$

and

$$t_{s,bus_{lo}} = \frac{\frac{nP_{ss}}{2m}\frac{L}{W_B} + \sum_{d=0}^{s-1} nP_{sd}\left\{\frac{L_f}{W_B} + \sum_{k=1}^{t-1} r_{s,sd}(k)\right\}}{\frac{nP_{ss}}{2m} + \sum_{d=0}^{s-1} nP_{sd}}. \tag{3.36}$$

Note that the mean segment bus access time is independent of whether the packet is a request or a reply packet.

In the first phase of arbitration, a segment bus is assigned to a request if the header flit of the request is in a flit buffer of the corresponding segment switch. Therefore, at the beginning of the second phase, a segment bus from the upper bus

group in segment s is free with probability $\left(1 - \sum_{k=1}^{t} u_s(k)/t\right)$ and a segment bus from the lower bus group is free with probability $\left(1 - \sum_{k=1}^{t} u_{s+1}(k)/t\right)$, on average. This probability is reflected in the inflated (larger than the nominal value) bus access time. Hence, the expected time interval between consecutive bus service completions in the upper bus group in segment s is given by

$$t_{s,bus_{up}access} = \frac{t_{s,bus_{up}}}{\frac{b}{2}(1 - \frac{1}{t}\sum_{k=1}^{t} u_s(k))}. \tag{3.37}$$

Here, $u_s(k)$ is the utilization of an upstream flit buffer in segment s by the k^{th} flit. Similarly, the expected time interval between consecutive bus service completions in the lower bus group in segment s is expressed as

$$t_{s,bus_{lo}access} = \frac{t_{s,bus_{lo}}}{\frac{b}{2}(1 - \frac{1}{t}\sum_{k=1}^{t} u_{s+1}(k))}. \tag{3.38}$$

Therefore, the mean time until the request of class s can get access to a segment bus while $\mathcal{K}_{s,p_{out}}$ other queue heads are occupied, is given by

$$T[\mathcal{K}_{s,p_{out}}] = \begin{cases} (\mathcal{K}_{s,p_{out}} - \frac{b}{2})t_{s,bus_{up}access} + \frac{t_{s,bus_{up}access}}{2} & \text{at the upper bus group,} \\ (\mathcal{K}_{s,p_{out}} - \frac{b}{2})t_{s,bus_{lo}access} + \frac{t_{s,bus_{lo}access}}{2} & \text{at the lower bus group.} \end{cases} \tag{3.39}$$

The second term in the RHS of each of the above expressions represents the mean residual bus access time. Equation (3.39) indicates that the bus server is load-

dependent; that is, the mean bus service time depends on $\mathcal{K}_{s,p_{out}}$.

Now consider the random variable $\mathcal{K}_{s,p_{out}}$ and assume that a request at the head of a processor output queue participates in the upper bus group arbitration. Recall that in each segment there is a total of $(2m+n)$ different request inputs for bus assignment because up to n inter-segment requests from processors, up to m requests from memory output queues for reply packets, and up to m outstanding memory requests chosen from memory arbiters can participate. However, the actual number of requests that can participate in bus assignment of the second phase is at most $(m+n)$ because up to n inter-segment requests from processors and up to m outstanding memory requests chosen by memory arbiters must originate at the same set of n processors. Hence, the value of $\mathcal{K}_{s,p_{out}}$ is bounded above by $(m+n-1)$.

Let $\eta_{s,p}$ denote the number of occupied processor output queue heads observed by a request that has just arrived at the head of a certain processor output queue in segment s (not in the set of $\eta_{s,p}$ queues). Similarly, we define $\eta_{s,m}$ as the number of occupied memory output queue heads observed by a request that has just arrived at the head of a processor output queue in segment s.

The random variable $\mathcal{K}_{s,p_{out}}$ can be expressed as the sum of non-negative integers $\eta_{s,p}$ and $\eta_{s,m}$, and is bounded above by $(m+n-1)$. The set of all feasible combinations of $\eta_{s,p}$ and $\eta_{s,m}$, is

$$\mathcal{K}_{s,p_{out}} = \{(\eta_{s,p}+\eta_{s,m}) \mid 0 \leq \eta_{s,p} \leq n-1 \text{ and } 0 \leq \eta_{s,m} \leq m\}.$$

Then, $\mathcal{K}_{s,p_{out}}$ has a multinomial distribution given as follows

$$P[\mathcal{K}_{s,p_{out}}] = \frac{(m+n-1)!p_{proc|bus_{eff}}^{\eta_{s,p}} p_{mem|bus}^{\eta_{s,m}} (1 - p_{proc|bus_{eff}} - p_{mem|bus})^{(m+n-1-\eta_{s,p}-\eta_{s,m})}}{(m+n-1-\eta_{s,p}-\eta_{s,m})!\eta_{s,p}!\eta_{s,m}!}. \tag{3.40}$$

The delay encountered by the header flit of a class s request at the head of a processor output queue until a segment bus is allocated to it is given by

$$s_{d|s,p_{out}} = \sum_{i=b/2}^{m+n-1} T[\mathcal{K}_{s,p_{out}} = i] \sum_{\mathcal{K}_{s,p_{out}}} P[\mathcal{K}_{s,p_{out}} = i]. \tag{3.41}$$

Now, we return to the calculation of unknown $u_{s,p_{out}}(1)$. $u_{s,p_{out}}(1)$ is simply the ratio of the residence time of a class s header flit at the head of a processor output queue to the round trip time of a class s request. Hence,

$$u_{s,p_{out}}(1) = \frac{r_{s,p_{out}}(1)}{\tau_p + R[s]} = \frac{R_{proc}[s]}{\tau_p + R[s]}. \tag{3.42}$$

The residence time of the header flit of class j at the head of a memory output queue in segment s will be denoted by $r_{s|j,m_{out}}(1)$ and will be discussed in the next section. Then, the mean utilization of a memory output queue in segment s by a header flit is

$$u_{s,m_{out}}(1) = \sum_{j=0}^{g-1} \frac{\frac{nP_{js}}{m} r_{s|j,m_{out}}(1)}{\tau_p + R[j]}. \tag{3.43}$$

Using the pipelining property of wormhole routing, the residence time of the k^{th} flit ($k > 1$) at the head of each output queue is the same as the residence time of the $(k-1)^{st}$ flit at the flit buffer of the first segment switch in the supposed path. Thus, $r_{i,p_{out}}(k)$, $k > 1$, is the same as the residence time of the $(k-1)^{st}$ flit of class i at the flit buffer of the first segment switch in the forward path. $r_{i,m_{out}}(k)$ is the same as the mean residence time of the $(k-1)^{st}$ flit at the flit buffer of the first segment switch in the return path. $r_{i,p_{out}}(k)$ and $r_{i,m_{out}}(k)$, for $k > 1$, are given as follows:

$$r_{i,p_{out}}(k) = \frac{P_{ii}\frac{L_f}{W_B} + \sum_{d=0}^{i-1} P_{id} r_{i,id}(k-1) + \sum_{d=i+1}^{g-1} P_{id} r_{i+1,id}(k-1)}{P_{ii} + \sum_{d=0}^{i-1} P_{id} + \sum_{d=i+1}^{g-1} P_{id}}$$

$$= P_{ii}\frac{L_f}{W_B} + \sum_{d=0}^{i-1 i-1} P_{id} r_{i,id}(k-1) + \sum_{d=i+1}^{g-1} P_{id} r_{i+1,id}(k-1) \qquad (3.44)$$

and

$$r_{i,m_{out}}(k) = P_{ii}\frac{L_f}{W_B} + \sum_{s=0}^{i-1} P_{si} r_{i,is}(k-1) + \sum_{s=i+1}^{g-1} P_{si} r_{i+1,is}(k-1). \qquad (3.45)$$

The utilizations $u_{i,p_{out}}(k)$ and $u_{i,m_{out}}(k)$ can be calculated easily as

$$u_{i,p_{out}}(k) = \frac{r_{i,p_{out}}(k)}{\tau_p + R[i]} \qquad (3.46)$$

and

$$u_{i,m_{out}}(k) = \sum_{s=0}^{g-1} \frac{\frac{nP_{si}}{m}\tau_{i,m_{out}}(k)}{\tau_p + R[s]}. \tag{3.47}$$

3.6 Residence Time in a Memory Module

The residence time in a memory module is the sum of mean memory input queue delay and mean memory output queue delay and is given by

$$R_{mem}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{d|s,mem_{in}} + R_{d|s,mem_{out}}) \qquad s = 0, \ldots, g-1.$$

3.6.1 Memory Input Queue Delays

Mean memory input queue delay, $R_{d|s,mem_{in}}$, includes the mean time a class s request spends in the input queue of the addressed target memory module in segment d ($w_{d|s,m_{in}}$) and the memory service time (τ_m). We assume that a request is queued for memory access only when its tail flit arrives at a memory input queue.

$$\begin{aligned} R_{d|s,mem_{in}} &= w_{d|s,m_{in}} + \tau_m \\ &= q_{d|s,m_{in}}\tau_m + u_{d|s,m}\gamma_m + \tau_m \end{aligned} \tag{3.48}$$

$w_{d|s,m_{in}}$ consists of the queueing delay in the memory input queue ($q_{d|s,m_{in}}\tau_m$) and the mean residual lifetime of a memory request in service ($u_{d|s,m}\gamma_m$). The mean memory input queue length, $q_{d|s,m_{in}}$, observed by a class s request packet when it

arrives at the memory input queue in segment d, is given by

$$q_{d|s,m_{in}} = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} w_{d|s',m_{in}}}{\tau_p + R[s']}. \tag{3.49}$$

In calculating the mean queue length, one should not count the contribution by the class s request that has just arrived. However, this fact can be ignored because the contribution is expected to be very small considering typical values for n and g. Notice that there exist g classes in the model and for each class n requests are generated on average. The probability that a memory module is busy serving a request when a class s packet arrives at the memory module in segment d is given by

$$u_{d|s,m} = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} \tau_m}{\tau_p + R[s']}. \tag{3.50}$$

We now turn our attention to the derivation of the mean residual lifetime of a memory request at a memory module in service. Recall that a memory request is queued up only when its tail flit arrives at the memory input queue. Hence, the residual lifetime of a memory request should be calculated as seen by the tail flit rather than as seen by the header flit. As before, we assume that the header flit is equally likely to arrive at a memory input queue at any point during the memory service time interval. Define t_e to be the amount of time which has already elapsed in serving a request when the header flit observed a memory request in service. For simplicity, each component in the calculation of the residual lifetime will be expressed in terms of the flit time ($\frac{L_f}{W_B}$).

Suppose that t_e is less than one flit time. In this case, the residual lifetime seen by the header flit would be $(\tau_m - \frac{L_f}{2W_B})$ on average. If a constant T is set to $\tau_m / \frac{L_f}{W_B}$, the residual lifetime can be expressed as $(T - \frac{1}{2})\frac{L_f}{W_B}$. However, the actual residual lifetime should be calculated as seen by the tail flit. The probability that the request in service observed by the header flit will still be in service when the tail flit arrives at the input queue is given by

$$\frac{T - \frac{1}{2} - (t-1)}{T - \frac{1}{2}}$$

where t is the number of flits per packet. Therefore, when $0 < t_e < \frac{L_f}{W_B}$, the mean residual lifetime seen by the tail flit is

$$(T - \frac{1}{2})\left(\frac{T - \frac{1}{2} - (t-1)}{T - \frac{1}{2}}\right)\left(\frac{L_f}{W_B}\right).$$

Similarly, for $\frac{L_f}{W_B} \leq t_e < \frac{2L_f}{W_B}$, the mean residual lifetime is given by

$$(T - \frac{1}{2} - 1)\left(\frac{T - \frac{1}{2} - 1 - (t-1)}{T - \frac{1}{2} - 1}\right)\left(\frac{L_f}{W_B}\right).$$

If $\frac{kL_f}{W_B} \leq t_e < \frac{(k+1)L_f}{W_B}$, the mean residual lifetime is expressed as

$$(T - \frac{1}{2} - k)\left(\frac{T - \frac{1}{2} - k - (t-1)}{T - \frac{1}{2} - k}\right)\left(\frac{L_f}{W_B}\right).$$

This procedure continues while the tail flit can observe the request in service. In other words, the tail flit can observe a request that is still in service only while

$$T - \frac{1}{2} - k - (t-1) > 0. \tag{3.51}$$

Conditioned on this, k is given by $T - t$ and is assumed to be a positive integer. Since the header flit is equally likely to arrive at a memory input queue at any point during the memory service time interval (τ_m), the mean residual lifetime of a memory request can be determined as

$$\begin{aligned}
&\frac{L_f/W_B}{\tau_m} \sum_{k=0}^{T-t} (T - \frac{1}{2} - k) \frac{\left(T - \frac{1}{2} - k - (t-1)\right)}{T - \frac{1}{2} - k} \left(\frac{L_f}{W_B}\right) \\
&= \frac{1}{\tau_m} \sum_{k=0}^{T-t} (T - t + \frac{1}{2} - k) \left(\frac{L_f}{W_B}\right)^2 \\
&= \frac{1}{\tau_m} \left[(T - t + \frac{1}{2})(T - t + 1) - \frac{(T-t)(T-t+1)}{2} \right] \left(\frac{L_f}{W_B}\right)^2 \\
&= \frac{(T-t+1)^2}{2\tau_m} \left(\frac{L_f}{W_B}\right)^2 \\
&= \frac{(\tau_m - \frac{L - L_f}{W_B})^2}{2\tau_m}.
\end{aligned} \tag{3.52}$$

Thus, the mean residual lifetime of a memory request in service, as seen by the tail flit, is given by

$$\gamma_m = \frac{(\tau_m - \frac{L - L_f}{W_B})^2}{2\tau_m}. \tag{3.53}$$

3.6.2 Memory Output Queue Delays

Mean memory output queue delay, $R_{d|s,mem_{out}}$, is the sum of the mean queueing delay, $w_{d|s,m_{out}}$, and the mean residence time of the header flit at the head of a memory output queue, $r_{d|s,m_{out}}(1)$. Thus,

$$R_{d|s,mem_{out}} = w_{d|s,m_{out}} + r_{d|s,m_{out}}(1). \tag{3.54}$$

$w_{d|s,m_{out}}$ is the time until the header flit of a reply packet of class s arrives at the head of a memory output queue in segment d.

As we discussed earlier in the case of $r_{s,p_{out}}(1)$, $r_{d|s,m_{out}}(1)$ includes the mean waiting time for the header flit of a class s packet at the head of the memory output queue to get access to a segment bus in segment d ($s_{d|s,m_{out}}$), the mean residual residence time for the tail flit in service ($\gamma_{d|s,m_{out}}$) (as the header flit arrives at the first segment switch in the return path), and a flit time for transferring the header flit to its first segment switch for an inter-segment access or to its processor for an intra-segment access.

The mean queueing delay is given by

$$\begin{aligned} w_{d|s,m_{out}} &= q_{d|s,m_{out}} \sum_{k=1}^{t} r_{d,m_{out}}(k) \\ &\quad + \sum_{k=1}^{t} u_{d|s,m_{out}}(k) \left(\frac{r_{d,m_{out}}(k)}{2} + \sum_{j=k+1}^{t} r_{d,m_{out}}(j) \right). \end{aligned} \tag{3.55}$$

The mean memory output queue length observed by the header flit of a class s reply packet when it arrives at a memory output queue in segment d is given by

$$q_{d|s,m_{out}} = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} w_{d|s',m_{out}}}{\tau_p + R[s']}. \tag{3.56}$$

As before, since the amount contributed to the queue length by a class s request that has just arrived is insignificant, it will be ignored. The mean residence time of the k^{th} flit of a reply packet at a memory output queue in segment d, $r_{d,m_{out}}(k)$ for $k > 1$, was discussed in Section 3.5. Here, we rewrite it for convenience as

$$r_{d,m_{out}}(k) = P_{dd}\frac{L_f}{W_B} + \sum_{s=0}^{d-1} P_{sd} r_{d,ds}(k-1) + \sum_{s=d+1}^{g-1} P_{sd} r_{d+1,ds}(k-1). \tag{3.57}$$

The mean residence time experienced by the header flit of a reply packet (of any class) at a memory output queue in segment d is expressed as

$$r_{d,m_{out}}(1) = \sum_{s=0}^{g-1} \frac{nP_{sd}}{m} s_{d|s,m_{out}} + \sum_{\substack{s=0 \\ s\neq d}}^{g-1} \frac{nP_{sd}}{m} \gamma_{d|s,m_{out}} + \frac{L_f}{W_B}. \tag{3.58}$$

In Equation (3.55), the second term shows the mean residual service time of a reply packet in service. The arriving packet at the memory output queue observes the k^{th} flit utilizing the memory output queue with probability

$$u_{d|s,m_{out}}(k) = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} r_{d|s',m_{out}}(k)}{\tau_p + R[s']}. \tag{3.59}$$

In this case, the mean residual residence time of that packet is

$$\frac{r_{d,m_{out}}(k)}{2} + \sum_{j=k+1}^{t} r_{d,m_{out}}(j).$$

The residence time of the header flit of a class s reply packet at the memory output queue in segment d, $r_{d|s,m_{out}}(1)$, is given by

$$r_{d|s,m_{out}}(1) = s_{d|s,m_{out}} + \gamma_{d|s,m_{out}} + \frac{L_f}{W_B}. \tag{3.60}$$

Calculation of $r_{d|s,m_{out}}(1)$ could be done in a very similar way as in Section 3.5, where the residence time of a header flit at a processor output queue was estimated. Similarly, we define $\theta_{d,p}$ as the number of occupied processor output queue heads observed by a reply packet when it arrives at the head of a memory output queue in segment d. We also define $\theta_{d,m}$ as the number of occupied memory output queue heads observed by a reply packet when it arrives at the head of a memory output queue in segment d.

The random variable $\mathcal{K}_{d,m_{out}}$ can be written as the sum of non-negative integers $\theta_{d,p}$ and $\theta_{d,m}$, and is bounded above by $(n+m-1)$. The set of all feasible values for $\mathcal{K}_{d,m_{out}}$ can be expressed as

$$\mathcal{K}_{d,m_{out}} = \{(\theta_{d,p} + \theta_{d,m}) \mid 0 \leq \theta_{d,p} \leq n \text{ and } 0 \leq \theta_{d,m} \leq m-1\}.$$

Then, $\mathcal{K}_{d,m_{out}}$ follows a multinomial distribution given by

$$P[\mathcal{K}_{d,m_{out}}] = \frac{(m+n-1)!p_{proc|bus_{eff}}^{\theta_{d,p}} p_{mem|bus}^{\theta_{d,m}} (1-p_{proc|bus_{eff}}-p_{mem|bus})^{(m+n-1-\theta_{d,p}-\theta_{d,m})}}{(m+n-1-\theta_{d,p}-\theta_{d,m})!\theta_{d,p}!\theta_{d,m}!}. \tag{3.61}$$

The delay, $s_{d|s,m_{out}}$, is given as follows

$$s_{d|s,m_{out}} = \sum_{i=b/2}^{m+n-1} T[\mathcal{K}_{d,m_{out}}=i] \sum_{\mathcal{K}_{d,m_{out}}} P[\mathcal{K}_{d,m_{out}}=i] \tag{3.62}$$

where

$$T[\mathcal{K}_{d,m_{out}}] = \begin{cases} (\mathcal{K}_{d,m_{out}} - \frac{b}{2}) t_{d,bus_{up}access} + \frac{t_{d,bus_{up}access}}{2} & \text{at the upper bus group,} \\ (\mathcal{K}_{d,m_{out}} - \frac{b}{2}) t_{d,bus_{lo}access} + \frac{t_{d,bus_{lo}access}}{2} & \text{at the lower bus group.} \end{cases}$$

The mean residual residence time, $\gamma_{d|s,m_{out}}$, is calculated in the same way as in the previous section and can be expressed as

$$\gamma_{d|s,m_{out}} = \begin{cases} u_d(t)\frac{r_d(t)}{2} & \text{if } d-s>0, \\ u_{d+1}(t)\frac{r_{d+1}(t)}{2} & \text{if } d-s<0. \end{cases} \tag{3.63}$$

Here, $\gamma_{d|s,m_{out}}$ becomes zero for $s=d$ as we discussed earlier.

CHAPTER 4

QUEUEING ANALYSIS FOR SYSTEMS WITH INFINITE FLIT BUFFERS

This chapter develops a performance analysis model for studying SMBS's with infinite flit buffers. We develop equations for mean residence times at three distinct infinite queueing centers: the segment switch, processor and memory module.

4.1 OVERVIEW OF THE MODEL

Model assumptions for this case are very similar to those stated in Chapter 3 for the single flit buffer model, with the exception of buffer size, of course. The assumptions are as follows:

1. Each processor generates request messages with exponential distribution of the inter-request interval with a mean of τ_p.

2. Read and write accesses in a memory module require the same service interval and take a constant time, τ_m.

3. A transient packet has priority over other packets for bus assignment. This assumption is based on the expectation that quickly delivered packets will reduce contention for network resources.

4. Arbitration time is included in the bus cycle time and a fair selection policy is utilized in each arbitration phase.

5. Bus service time is deterministic and is equal to a bus cycle time which is also equal to the flit time.

6. The queueing discipline at each memory queue is first-come first-served (FCFS). In the case when requests arrive simultaneously, these requests are served in random order.

In addition to the above mentioned assumptions, we assume that the capacity of a flit buffer is unlimited. The flit buffer's queueing discipline is FCFS as well. Therefore, the whole packet can be buffered at one flit buffer when a header flit is blocked; unlike the case of a finite flit buffer when the trailing flits are blocked in place and occupy simultaneously several segment switches. Under the condition of unlimited flit buffer capacity, a header flit can go forward without the entire packet being buffered as soon as the next segment bus and segment switch are available. This guarantees that a network with infinite flit buffers will, in general, offer less network latency than a network with finite flit buffers. We assume wormhole routing with infinite flit buffers throughout this chapter.

We maintain the assumption that memory requests within a segment are uniformly distributed and segment referencing is distributed arbitrarily. Each memory module is assumed to have an input buffer and an output buffer so that the memory module may serve requests continuously without having to wait to transmit reply packets back to their requesting processors. The sizes of memory input and output buffers are unlimited. A reply packet is assumed to be immediately consumed as soon as it arrives at its requesting processor.

Unlike the single flit buffer model, the infinite buffer model does not consider the mean residual residence time of a tail flit and the blocking delay because a packet arriving at a segment switch will not experience shortage of buffer space.

Consider a class s request for a memory module in segment d. A processor will generate a class s request which will travel (in a forward path) from s to d and will receive a reply packet, of the same class, from the targeted memory module in segment d (in a return path). The residence time experienced by the request will be the sum of mean residence time (queueing and service) in the processor, in the network and in the memory module. Hence, the average response time of a class s request becomes

$$R[s] = R_{proc}[s] + R_{network}[s] + R_{mem}[s] \qquad s = 0, \ldots, g-1. \tag{4.1}$$

$R_{proc}[s]$ denotes the mean time that the header flit of a class s packet spends at the head of a processor output queue. This is the sum of the mean waiting time for a segment bus and the time for transferring the header flit to the first segment switch (in the case of an inter-segment access) or the time for transferring the header flit on a segment bus to a local target memory module (in the case of an intra-segment access).

$R_{network}[s]$ is the weighted sum of the mean residence times in the forward path (for a request packet) and in the return path (for a reply). $R_{network}[s]$ is given by

$$R_{network}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{sd,network} + R_{ds,network}) \qquad s = 0, \ldots, g-1, \qquad (4.2)$$

where $\sum_{d=0}^{g-1} P_{sd} = 1$. $R_{sd,network}$ denotes the average residence time in the network of a class s request packet from the first segment switch in the forward path to the target memory in segment d. $R_{ds,network}$ denotes the average residence time in the network of a class s reply packet from the first segment switch in the return path to the requesting processor.

The residence time at each segment switch consists of the mean queueing delay at a flit buffer queue and the time for transferring the header flit to the next switch or to the target memory module (for the final segment switch in the forward path) or to the original requesting processor (for the final segment switch in the return path). Once the header flit arrives at a target memory module in the forward path, the remaining flits will follow (catch-up) and reach the target memory. Similarly, once the header flit arrives at the requesting processor in the return path, the remaining flits will follow (catch-up) and reach the processor. This is known as catch-up time.

$R_{mem}[s]$ is the residence time in a memory module and is given by

$$R_{mem}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{d|s,mem_{in}} + R_{d|s,mem_{out}}) \qquad s = 0, \ldots, g-1. \qquad (4.3)$$

$R_{mem}[s]$ is exactly the same as in the single flit buffer model case. $R_{d|s,mem_{in}}$ is the sum of the mean delay for a class s request packet experienced in the memory input queue in segment d and the memory service time, τ_m. $R_{d|s,mem_{out}}$ is the average

delay that the reply packet of class s spends in the memory output queue. In the case of an inter-segment access, this consists of the mean queueing delay at the memory output queue, the mean waiting time for a segment bus at the head of the memory output queue and the time for transferring the header flit to the first segment switch. In the case of an intra-segment access, $R_{d|s,mem_{out}}$ consists of the mean queueing delay, the segment bus waiting time and the time for transferring the reply header flit back to its local processor.

We will also utilize the same performance metrics in this case, as those utilized in the single flit buffer model: the overall mean response time (R) and processing efficiency (ρ_{proc}). These are given by

$$R = \frac{1}{g}\sum_{s=0}^{g-1} R[s] \tag{4.4}$$

and

$$\rho_{proc} = \frac{1}{g}\sum_{s=0}^{g-1} \rho_{proc}[s], \tag{4.5}$$

where

$$\rho_{proc}[s] = \frac{\tau_p}{\tau_p + R[s]}.$$

4.2 Residence Time in a Processor

Residence time of a header flit in a processor output queue, basically, consists of the following two elements:

1. The mean waiting time for a header flit of a class s request directed to d in order to compete for a segment bus at a processor output queue in segment s ($s_{d|s,p_{out}}$).

2. The flit time for transferring the header flit to the first segment switch in the forward path to d, for an inter-segment access, or the time for transferring the header flit to its addressed local memory module, for an intra-segment access ($\frac{L_f}{W_B}$).

Unlike the case of the single flit buffer model, a header flit does not observe any residual residence time of a tail flit in service at the next segment switch because the size of a flit buffer is unbounded. Hence, the mean residence time of the header flit of class s at the processor output queue is given by

$$R_{proc}[s] = \sum_{d=0}^{g-1} P_{sd} s_{d|s,p_{out}} + \frac{L_f}{W_B} \qquad s = 0, \ldots, g-1. \tag{4.6}$$

Consider a class s request. As we discussed in the single flit buffer model in Chapter 3, we assume that a request packet observes $\mathcal{K}_{s,p_{out}}$ other requests occupying the heads of their respective queues when the request packet arrives at the head of its processor output queue, where $\mathcal{K}_{s,p_{out}}$ is an integer random variable. We follow the same analysis as in the single flit buffer model of Chapter 3. If $\mathcal{K}_{s,p_{out}}$ is smaller than the number of available free segment buses in a bus group, the request doesn't need to wait for a segment bus assignment. However, if $\mathcal{K}_{s,p_{out}}$ is equal to or greater than the number of free segment buses, the request must wait until a free segment

bus is available. Suppose that i is the number of free buses observed by the header flit when it arrives at the head of a processor output queue. If $\mathcal{K}_{s,p_{out}}$ is greater than i, the request will wait $(\mathcal{K}_{s,p_{out}} - i)$ service completions before it can get access to a segment bus. This is because when a bus becomes free, the next request to be granted the bus is chosen according to a FCFS policy.

The mean segment bus access time is the time for transferring a request or a reply packet on a segment bus. The segment bus access time will be simply L/W_B because here we do not have to consider the blocking effect caused by a finite flit buffer capacity.

In the first phase of arbitration, the segment bus is assigned the header flit of a request in the corresponding segment switch if such request exists. Therefore, at the beginning of the second phase, the probability that a segment bus (supposedly, of the upper bus group in segment i) is busy serving a transient packet, denoted by $u_{i,bus}$, is given by

$$u_{i,bus} = \sum_{s=0}^{g-1} \sum_{d \in D^{+}_{i,sd}} \frac{P_{sd}\frac{L}{W_B}}{\tau_p + R[s]} + \sum_{s=0}^{g-1} \sum_{d \in D^{-}_{i,sd}} \frac{P_{sd}\frac{L}{W_B}}{\tau_p + R[s]} \tag{4.7}$$

where

$$D^{+}_{i,sd} \equiv \{d \mid \text{packets traveling from } s \text{ to } d \text{ will visit upstream flit buffer } i\}$$

$$D^{-}_{i,sd} \equiv \{d \mid \text{packets traveling from } s \text{ to } d \text{ will visit downstream flit buffer } i\}.$$

This probability reflects a larger bus access time than the nominal value of $\frac{L}{W_B}$. Hence, the expected time interval between consecutive bus service completions in

the upper bus group of segment s is given by

$$t_{s,bus_{up}access} = \frac{\frac{L}{W_B}}{\frac{b}{2}(1 - u_{s,bus})}. \tag{4.8}$$

Similarly, the expected time interval between consecutive bus service completions in the lower bus group of segment s is expressed as

$$t_{s,bus_{lo}access} = \frac{\frac{L}{W_B}}{\frac{b}{2}(1 - u_{s+1,bus})}. \tag{4.9}$$

Therefore, the mean time until the class s request can get access to a segment bus while $\mathcal{K}_{s,p_{out}}$ other queue heads are occupied, is given by

$$T[\mathcal{K}_{s,p_{out}}] = \begin{cases} (\mathcal{K}_{s,p_{out}} - \frac{b}{2})t_{s,bus_{up}access} + \frac{t_{s,bus_{up}access}}{2} & \text{at the upper bus group,} \\ (\mathcal{K}_{s,p_{out}} - \frac{b}{2})t_{s,bus_{lo}access} + \frac{t_{s,bus_{lo}access}}{2} & \text{at the lower bus group.} \end{cases} \tag{4.10}$$

The equation shows that the bus server is load-dependent; that is, the mean time depends on $\mathcal{K}_{s,p_{out}}$.

Now consider the integer random variable $\mathcal{K}_{s,p_{out}}$. We use the same variable definitions and notation (without any modifications) as those defined in the single flit buffer model. Thus, $\mathcal{K}_{s,p_{out}}$ is given as

$$P[\mathcal{K}_{s,p_{out}}] = \frac{(m+n-1)!p_{proc|bus_{eff}}^{\eta_{s,p}} p_{mem|bus}^{\eta_{s,m}} (1 - p_{proc|bus_{eff}} - p_{mem|bus})^{(m+n-1-\eta_{s,p}-\eta_{s,m})}}{(m+n-1-\eta_{s,p}-\eta_{s,m})!\eta_{s,p}!\eta_{s,m}!}. \tag{4.11}$$

The delay experienced by the header flit of a class s packet at the head of the processor output queue, until a segment bus is allocated to it, is given by

$$s_{d|s,p_{out}} = \sum_{i=b/2}^{m+n-1} T[\mathcal{K}_{s,p_{out}} = i] \sum_{\mathcal{K}_{s,p_{out}}} P[\mathcal{K}_{s,p_{out}} = i]. \tag{4.12}$$

4.3 Residence Time in the Network

The residence time in the network is the weighted sum of the mean residence times in the forward path for a request packet and in the return path for a reply packet, as given below:

$$R_{network}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{sd,network} + R_{ds,network}) \qquad s = 0, \ldots, g-1, \tag{4.13}$$

where $\sum_{d=0}^{g-1} P_{sd} = 1$.

$R_{sd,network}$ corresponds to the average residence time in the network of a class s request packet from the first segment switch in the forward path to an addressed target memory module in segment d. Similarly, $R_{ds,network}$ corresponds to the average residence time in the network of a class s reply packet from the first segment switch in the return path to the original requesting processor.

The mean residence time for a class s packet in the forward path (or in the return path) consists of the following two elements:

1. The sum of the mean residence times of the header flit of a class s packet at all segment switches in the forward path from s to d, $\sum_i r_{i,sd}(1)$ (or in the return path from d to s, $\sum_i r_{i,ds}(1)$).

2. The catch-up time, $\frac{L-L_f}{W_B}$, at the addressed target memory module in the case of the forward path or at the requesting processor in the case of the return path.

Therefore, the mean residence time in the forward path can be written as

$$R_{sd,network} = \begin{cases} \sum_{i=s+1}^{d} r_{i,sd}(1) + \frac{L-L_f}{W_B} & \text{if } d-s>0, \\ \sum_{i=d+1}^{s} r_{i,sd}(1) + \frac{L-L_f}{W_B} & \text{if } d-s<0. \end{cases} \tag{4.14}$$

In the return path, the mean residence time is given by

$$R_{ds,network} = \begin{cases} \sum_{i=s+1}^{d} r_{i,ds}(1) + \frac{L-L_f}{W_B} & \text{if } d-s>0, \\ \sum_{i=d+1}^{s} r_{i,ds}(1) + \frac{L-L_f}{W_B} & \text{if } d-s<0. \end{cases} \tag{4.15}$$

In the case of an intra-segment access ($d = s$), $r_{i,sd}(1)$ and $r_{i,ds}(1)$ are equal to zero.

Suppose that a class s packet waits for a segment bus at segment switch i in the upstream forward path from s to d. We call such packet the *tagged packet*. The mean residence time of the header flit of the tagged packet at segment switch i in the forward path, $r_{i,sd}(1)$, consists of the following three elements:

1. The mean waiting time for packets which were queued earlier at the flit buffer queue.

2. The mean residual service time for a packet if the tagged header flit finds the segment bus busy.

3. The time for transferring the tagged header flit to the next segment switch: a flit time ($\frac{L_f}{W_B}$).

Again, please observe that in this case we do not include the mean residual residence time of the tail flit of a previous packet because the flit buffer is assumed unbounded. Thus, the residence time, $r_{i,sd}(1)$, in the upstream forward path is given by

$$r_{i,sd}(1) = q_{i,sw}\frac{L}{W_B} + p_{bus_i,busy}\frac{L}{2W_B} + \frac{L_f}{W_B}. \tag{4.16}$$

$r_{i,ds}(1)$ in the upstream return path has the same expression as that of $r_{i,sd}(1)$ in the upstream forward path.

The mean queue length seen by the arriving tagged header flit at the flit buffer queue in segment i, denoted by $q_{i,sw}$, is given by

$$q_{i,sw} = \sum_{s=0}^{g-1}\sum_{d\in D^{+}_{i,sd}}\frac{P_{sd}r_{i,sd}(1)}{\tau_p + R[s]} + \sum_{s=0}^{g-1}\sum_{d\in D^{-}_{i,sd}}\frac{P_{sd}r_{i,ds}(1)}{\tau_p + R[s]} \tag{4.17}$$

where

$$D^{+}_{i,sd} \equiv \{d \mid \text{packets from } s \text{ to } d \text{ will visit upstream flit buffer } i\}$$

$$D^{-}_{i,sd} \equiv \{d \mid \text{packets from } s \text{ to } d \text{ will visit downstream flit buffer } i\}.$$

Once the tagged header flit arrives at the head of the flit buffer queue, the header flit may wait for a segment bus. When the segment bus is occupied by a local processor or a memory module, the tagged header flit must wait until the segment bus is released. In Equation (4.16), $\frac{L}{2W_B}$ is the mean residual service time of a packet. $p_{bus_i,busy}$ denotes the probability that the tagged flit observes segment bus i busy serving a request from a local processor or a memory module.

Let the integer random variable Φ_s denote the number of requests that will participate in bus assignment of the second arbitration phase at segment s. Recall that in each segment the actual number of requests that participate in bus assignment is at most $n + m$: up to n inter-segment requests from processors and up to m outstanding memory requests chosen from memory arbiters. Hence, Φ_s is bounded above by $n + m$.

Let $\phi_{s,p}$ and $\phi_{s,m}$ denote the number of processors and memory modules participating in bus arbitration, respectively. Φ_s can be expressed as the sum of the non-negative integers $\phi_{s,p}$ and $\phi_{s,m}$ whose sum is bounded above by $n + m$. The set of all feasible combinations of $\phi_{s,p}$ and $\phi_{s,m}$, is given as $\Phi_s = \{(\phi_{s,p} + \phi_{s,m}) \mid 0 \leq \phi_{s,p} \leq n \text{ and } 0 \leq \phi_{s,m} \leq m\}$. If we assume that the trials performed by processors and memory modules are independent Bernoulli trials, then Φ_s has a multinomial distribution given as follows

$$P[\Phi_s] = \frac{(n+m)! p_{proc|bus_{eff}}^{\phi_{s,p}} p_{mem|bus}^{\phi_{s,m}} (1 - p_{proc|bus_{eff}} - p_{mem|bus})^{(n+m-\phi_{s,p}-\phi_{s,m})}}{(n+m-\phi_{s,p}-\phi_{s,m})! \phi_{s,p}! \phi_{s,m}!} \tag{4.18}$$

where the probabilities $p_{proc|bus_{eff}}$ and $p_{mem|bus}$ are the same as those discussed in Chapter 3.

If i requests, $i \leq \frac{b}{2}$, participate in bus assignment of the second arbitration phase, the tagged header flit will observe the corresponding segment bus being occupied with probability $i/\frac{b}{2}$, on average. If the number of requests is greater than the number of segment buses per bus group ($b/2$), the tagged header flit will always find the corresponding segment bus busy. Therefore, the probability that the header flit observes the corresponding segment bus busy is estimated as

$$p_{bus_s,busy} = \sum_{i=1}^{\frac{b}{2}-1} P[\Phi_s = i]\frac{i}{\frac{b}{2}} + \sum_{i=\frac{b}{2}}^{n+m} P[\Phi_s = i]. \tag{4.19}$$

Similarly, the residence time, $r_{i,sd}(1)$ in the downstream forward path (or $r_{i,ds}(1)$ in the downstream return path) is given by

$$r_{i,sd}(1) = q_{i,sw}\frac{L}{W_B} + p_{bus_{i-1},busy}\frac{L}{2W_B} + \frac{L_f}{W_B}. \tag{4.20}$$

As before, we state $r_{i,sd}(1)$ without using an explicit indication as to whether the path taken is an upstream or a downstream path.

4.4 Residence Time in a Memory Module

The residence time in a memory module is given by

$$R_{mem}[s] = \sum_{d=0}^{g-1} P_{sd}(R_{d|s,mem_{in}} + R_{d|s,mem_{out}}) \qquad s = 0, \ldots, g-1. \tag{4.21}$$

The residence time in a memory module is the same as that derived in the single flit buffer case, except that the residence time for the header flit of a reply packet at a memory output queue does not include the tail flit residual residence time. Mean memory input queue delay, $R_{d|s,mem_{in}}$, includes the mean time that a class s request spends in the input queue of the addressed target memory module in segment d ($w_{d|s,m_{in}}$) and the memory service time (τ_m).

As we did in the single flit buffer case, we assume here that a request is queued for memory access only when its tail flit arrives at the memory input queue. Mean memory input queue delay, $R_{d|s,mem_{in}}$, is given by

$$\begin{aligned} R_{d|s,mem_{in}} &= w_{d|s,m_{in}} + \tau_m \\ &= q_{d|s,m_{in}}\tau_m + u_{d|s,m}\gamma_m + \tau_m. \end{aligned} \tag{4.22}$$

$w_{d|s,m_{in}}$ consists of the queueing delay in the memory input queue ($q_{d|s,m_{in}}\tau_m$) and the mean residual lifetime of a memory request in service ($u_{d|s,m}\gamma_m$). The mean memory input queue length, $q_{d|s,m_{in}}$, observed by a class s request packet when it arrives at a memory input queue in segment d is given by

$$q_{d|s,m_{in}} = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} w_{d|s',m_{in}}}{\tau_p + R[s']}. \tag{4.23}$$

As in Chapter 3, the mean queue length does not include the amount contributed by the class s request that has just arrived. The probability that the memory is busy serving a request when a class s packet arrives at a memory module in segment d,

is expressed as

$$u_{d|s,m} = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m}\tau_m}{\tau_p + R[s']}. \tag{4.24}$$

The mean residual lifetime of a memory request in service, γ_m (derived in Chapter 3), is given by

$$\gamma_m = \frac{(\tau_m - \frac{L-L_f}{W_B})^2}{2\tau_m}. \tag{4.25}$$

Mean memory output queue delay, $R_{d|s,mem_{out}}$, is the sum of the mean queueing delay ($w_{d|s,m_{out}}$) and the mean residence time of the header flit at the head of the memory output queue ($r_{d|s,m_{out}}(1)$). Thus,

$$R_{d|s,mem_{out}} = w_{d|s,m_{out}} + r_{d|s,m_{out}}(1). \tag{4.26}$$

The mean queueing delay is given by

$$\begin{aligned} w_{d|s,m_{out}} &= q_{d|s,m_{out}} \sum_{k=1}^{t} r_{d,m_{out}}(k) \\ &\quad + \sum_{k=1}^{t} u_{d|s,m_{out}}(k) \left(\frac{r_{d,m_{out}}(k)}{2} + \sum_{j=k+1}^{t} r_{d,m_{out}}(j) \right). \end{aligned} \tag{4.27}$$

The mean memory output queue length which is observed by the header flit of a class s reply packet when it arrives at a memory output queue in segment d is given

by

$$q_{d|s,m_{out}} = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} w_{d|s',m_{out}}}{\tau_p + R[s']}. \tag{4.28}$$

The mean residence time of the k^{th} flit of a reply packet at a memory output in segment d, $r_{d,m_{out}}(k)$ for $k > 1$ is given by

$$r_{d,m_{out}}(k) = P_{dd}\frac{L_f}{W_B} + \sum_{s=0}^{d-1} P_{sd} r_{d,ds}(k-1) + \sum_{s=d+1}^{g-1} P_{sd} r_{d+1,ds}(k-1). \tag{4.29}$$

The mean residence time of the header flit of a reply packet at a memory output queue in segment d is expressed as

$$r_{d,m_{out}}(1) = \sum_{s=0}^{g-1} \frac{nP_{sd}}{m} s_{d|s,m_{out}} + \frac{L_f}{W_B}. \tag{4.30}$$

In Equation (4.27), the second term shows the mean residual service time of a reply packet in service. The arriving packet at the memory output queue in segment d observes the k^{th} flit utilizing the memory output queue with probability

$$u_{d|s,m_{out}}(k) = \sum_{s'=0}^{g-1} \frac{\frac{nP_{s'd}}{m} r_{d|s',m_{out}}(k)}{\tau_p + R[s']}. \tag{4.31}$$

In this case, the mean residual residence time of the packet in service at a segment bus in segment d is

$$\frac{r_{d,m_{out}}(k)}{2} + \sum_{j=k+1}^{t} r_{d,m_{out}}(j).$$

The residence time of the header flit of a class s reply packet at a memory output queue in segment d, $r_{d|s,m_{out}}(1)$, is given by

$$r_{d|s,m_{out}}(1) = s_{d|s,m_{out}} + \frac{L_f}{W_B}. \tag{4.32}$$

$r_{d|s,m_{out}}(1)$ includes the mean waiting time for the header flit of a class s packet to get access to a segment bus in segment d ($s_{d|s,m_{out}}$) and the flit time, for transferring the header flit to its first segment switch for an inter-segment access, or to the requesting processor for an intra-segment access.

Let $\theta_{d,p}$ denote the number of occupied processor output queue heads observed by a reply packet when it arrives at the head of a memory output queue in segment d. Let $\theta_{d,m}$ denote the number of occupied memory output queue heads observed by a reply packet when it arrives at the head of a memory output queue in segment d.

The number $\mathcal{K}_{d,m_{out}}$ can be written as the sum of non-negative integers $\theta_{d,p}$ and $\theta_{d,m}$ whose sum is bounded above by $(n+m-1)$. The set of all feasible values for $\mathcal{K}_{d,m_{out}}$ is given by

$$\mathcal{K}_{d,m_{out}} = \{(\theta_{d,p} + \theta_{d,m}) \mid 0 \leq \theta_{d,p} \leq n \text{ and } 0 \leq \theta_{d,m} \leq m-1\}.$$

Then, $\mathcal{K}_{d,m_{out}}$ follows a multinomial distribution given by

$$P[\mathcal{K}_{d,m_{out}}] = \frac{(m+n-1)!p_{proc|bus_{eff}}^{\theta_{d,p}} p_{mem|bus}^{\theta_{d,m}}(1-p_{proc|bus_{eff}}-p_{mem|bus})^{(m+n-1-\theta_{d,p}-\theta_{d,m})}}{(m+n-1-\theta_{d,p}-\theta_{d,m})!\theta_{d,p}!\theta_{d,m}!}. \tag{4.33}$$

The delay, $s_{d|s,m_{out}}$, is given as follows

$$s_{d|s,m_{out}} = \sum_{i=b/2}^{m+n-1} T[\mathcal{K}_{d,m_{out}}=i] \sum_{\mathcal{K}_{d,m_{out}}} P[\mathcal{K}_{d,m_{out}}=i] \tag{4.34}$$

where

$$T[\mathcal{K}_{d,m_{out}}] = \begin{cases} (\mathcal{K}_{d,m_{out}} - \frac{b}{2})t_{d,bus_{up}access} + \frac{t_{d,bus_{up}access}}{2} & \text{at the upper bus group,} \\ (\mathcal{K}_{d,m_{out}} - \frac{b}{2})t_{d,bus_{lo}access} + \frac{t_{d,bus_{lo}access}}{2} & \text{at the lower bus group.} \end{cases}$$

CHAPTER 5

NUMERICAL RESULTS AND DISCUSSION

In this chapter, we verify the correctness of our analytical models and obtain numerical data to study SMBS performance. Model input parameter values used in our experiments are presented. We use event-driven simulations to verify the correctness of our analytical models. We will show that the results obtained from analytical models are very close to those obtained from simulations.

We evaluate the performance of the SMBS under various configurations (various system sizes, segment configurations, flit buffer sizes and connection topologies) and under different workloads (varying memory request rates, number of flits and reference patterns). We study the performance under uniform memory reference and then examine the impact of memory reference locality on scalability for both the single flit buffer and infinite flit buffer models.

We address how the performance changes with different flit buffer sizes: namely single flit buffers, packet-sized flit buffers and infinite flit buffers, and also study the performance of two SMBS configurations: a ring connection of segments with infinite flit buffers and a linear connection of segments with single flit buffers and infinite flit buffers. Performance analysis will show that network latency is more influenced by bus contention than network distance. Sets of data obtained from both the packet-sized flit buffer model and the ring connection model are based on event-driven simulations.

Finally, we study the performance impact of the number of flits per packet and the number of segment buses per segment. Our results will show that the SMBS favors small number of flits per packet and wide bus width.

5.1 Model Input Parameters

Model parameters which have to be specified numerically can be classified into two basic categories: those that describe a particular system configuration and those that describe expected workloads. When we model a design that has not been built, parameter values will have to be estimated. In particular, determining workload parameter values properly is a difficult part of the analysis process. System configuration and workload parameters used in our model are listed in Table 3.1 in Chapter 3.

For our purposes here, a baseline system is configured as an SMBS with a linear connection of segments. Each segment is composed of 10 processors ($n = 10$), 10 memory modules ($m = 10$) and 10 segment buses ($b = 10$). For both the single flit buffer and infinite flit buffer cases, the baseline system is studied. To examine the impact of increasing system size on system scalability, we increase the number of segments in the SMBS for two different segment configurations: one for $n = 10$, $m = 10$ and $b = 10$, and the other for $n = 20$, $m = 20$ and $b = 10$. For studying the effect of the number of segment buses per segment on performance, we vary the number of segment buses in a segment from 6 to 12.

The flit time of $\frac{L_f}{W_B}$ will be used as our basic timing unit. Thus, we normalize $\frac{L_f}{W_B}$ to one unit of time, where L_f is the length of a flit and W_B is the segment bus

bandwidth. The mean memory request interval (τ_p) and memory service time (τ_m) will be in units of flit time. Considering a 32-bit wide segment bus and the fact that message length (packet length) is expected to be about 100 bits on average in a shared memory multiprocessor [46], we assume basic packet length to be three flits ($t = 3$). However, to study the effect of number of flits per packet, the number of flits per packet will be varied from 3 to 5.

The mean time of the request interval, τ_p, is varied from 30 to 200 time units. Values of τ_p higher than 200 show very little performance improvement. For values of $\tau_p \leq 30$ time units, some segment buses, most likely in middle segments, are nearly saturated particularly in experiments dealing with a baseline system with finite flit buffers. For such low values of memory request intervals, our model does not show a reasonable evaluation functionality. However, through extensive simulations, the given range for τ_p is shown to be reasonable enough to study the performance of our model. The mean memory service time, τ_m, is set to 4 units of time.

5.2 Validation of the Analytical Results

We used event-driven simulations to verify the correctness of our analytical models. The simulations modeled the actual behavior of the SMBS for both the single flit buffer and infinite flit buffer cases, without any simplifying assumptions. The simulator implemented wormhole routing and bus assignment priority of a transient packet exactly. Our simulation program (about 3,700 lines long) was written in the C programming language. It was driven by the multiplicative congruential random number generator [47, 48].

The chosen 95% confidence interval was observed to be within 10% of the mean; that is, if the confidence interval half width was within 10% of the mean, the desired accuracy was considered satisfactory. Simulation estimates were derived by using the method of batch means. Simulation results were finally obtained by averaging the results of ten batch simulations. Each processor generated 1,000 memory requests in each batch. Considering 10 indistinguishable processors in the basic segment configuration, each batch is equivalent to the case obtained by generating 10,000 memory requests of each class.

To deal with the start-up transient of the simulation, we truncated the first 10% of memory requests. This was observed to be long enough to make any start-up transient negligible. In terms of execution time, a simulation estimate of a baseline system requires about twenty hours of CPU time on a Sun SPARC-5 computer; however, numerical evaluations based on our analytical model took less than ten minutes each.

Simulation results are presented along with analytical results in Tables 5.1 – 5.4. Tables 5.1 and 5.2 present comparisons of analytical performance measures with simulation results for SMBS's of two different sizes with single flit buffers. For the case of infinite flit buffers, the results are presented in Tables 5.3 and 5.4 for two different sizes.

As can be seen, the differences between simulation results and analytical results are insignificant for both the single flit buffer and infinite flit buffer cases. Both are shown for $\tau_m = 4$ and $t = 3$, and two different system sizes: $g = 5$ and $g = 7$. In

Table 5.1: Comparison of performance with simulation results: SMBS with single flit buffers, $n = 10$, $m = 10$, $b = 10$, $g = 5$, $\tau_m = 4$ and $t = 3$.

Request interval, τ_p	Processing efficiency		Response time	
	Analysis	Simulation	Analysis	Simulation
40	72.226	72.472 ± .030	15.381	15.193 ± .023
50	76.981	77.147 ± .021	14.950	14.812 ± .018
70	82.885	82.923 ± .017	14.453	14.416 ± .017
100	87.658	87.604 ± .011	14.078	14.150 ± .015
150	91.582	91.498 ± .009	13.786	13.937 ± .017
200	93.615	93.524 ± .006	13.640	13.849 ± .014

Table 5.2: Comparison of performance with simulation results: SMBS with single flit buffers, $n = 10$, $m = 10$, $b = 10$, $g = 7$, $\tau_m = 4$ and $t = 3$.

Request interval, τ_p	Processing efficiency		Response time	
	Analysis	Simulation	Analysis	Simulation
40	67.668	67.612 ± .051	19.111	19.161 ± .045
50	73.466	73.663 ± .035	18.058	17.876 ± .032
70	80.511	80.587 ± .031	16.944	16.862 ± .033
100	86.079	86.027 ± .011	16.172	16.242 ± .014
150	90.575	90.450 ± .008	15.608	15.837 ± .015
200	92.877	92.744 ± .003	15.338	15.647 ± .008

Table 5.3: Comparison of performance with simulation results: SMBS with infinite flit buffers, $n = 10$, $m = 10$, $b = 10$, $g = 5$, $\tau_m = 4$ and $t = 3$.

Request interval, τ_p	Processing efficiency		Response time	
	Analysis	Simulation	Analysis	Simulation
30	66.877	67.011 ± .016	14.857	14.768 ± .011
40	73.359	73.505 ± .012	14.525	14.417 ± .009
50	77.754	77.878 ± .011	14.304	14.202 ± .009
70	83.304	83.363 ± .008	14.028	13.969 ± .008
100	87.870	87.884 ± .009	13.803	13.785 ± .012
150	91.678	91.655 ± .008	13.616	13.655 ± .014
200	93.668	93.648 ± .003	13.518	13.563 ± .007

Table 5.4: Comparison of performance with simulation results: SMBS with infinite flit buffers, $n = 10$, $m = 10$, $b = 10$, $g = 7$, $\tau_m = 4$ and $t = 3$.

Request interval, τ_p	Processing efficiency		Response time	
	Analysis	Simulation	Analysis	Simulation
30	63.454	63.634 ± .023	17.277	17.144 ± .017
40	70.472	70.813 ± .020	16.759	16.486 ± .016
50	75.295	75.621 ± .018	16.404	16.118 ± .016
70	81.440	81.682 ± .011	15.952	15.697 ± .011
100	86.521	86.649 ± .007	15.577	15.407 ± .010
150	90.763	90.805 ± .004	15.264	15.188 ± .007
200	92.980	92.989 ± .004	15.099	15.078 ± .011

all these cases, the analytical and simulation results match very well and are within 5% of one another.

5.3 Numerical Results

To examine how SMBS's scale with increasing system size, we evaluated processing efficiency of the systems with single flit buffers under uniform memory reference as a function of the number of segments (g) for different request rates ($1/\tau_p$). The results are shown in Figures 5.1 and 5.2 for two different segment configurations: one for $n = 10$, $m = 10$ and $b = 10$, and the other for $n = 20$, $m = 20$ and $b = 10$, respectively. Both figures show that the performance in both cases is poor as processing efficiency decreases rapidly with increasing number of segments (system size) at moderate or high request rates ($1/\tau_p \geq 0.02$ in Figure 5.1 and $1/\tau_p \geq 0.01$ in Figure 5.2). We observe that the low processing efficiency is primarily due to the increase in bus contention as request rate increases, as we can easily imagine particularly since buffer size is finite.

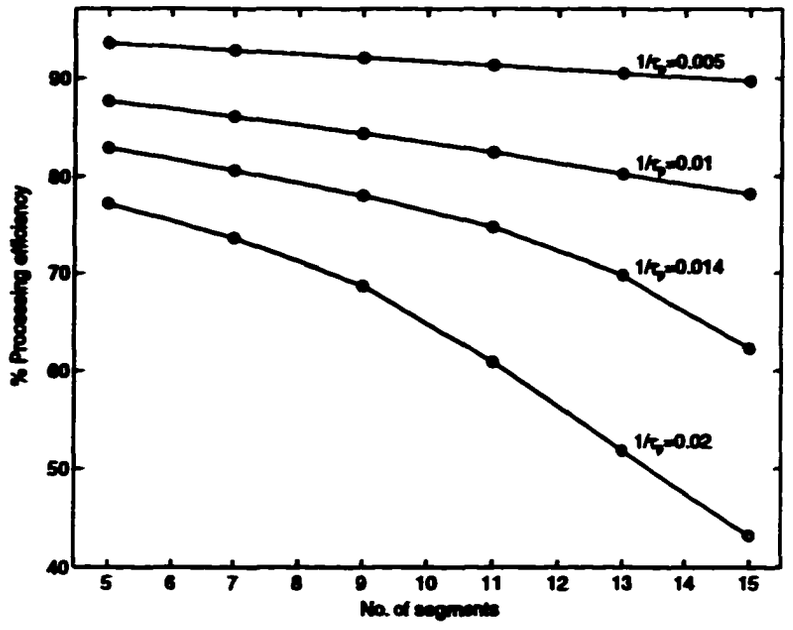


Figure 5.1: Processing efficiency of single flit buffer model under uniform memory reference: $n = 10$, $m = 10$, $b = 10$, $\tau_m = 4$ and $t = 3$.

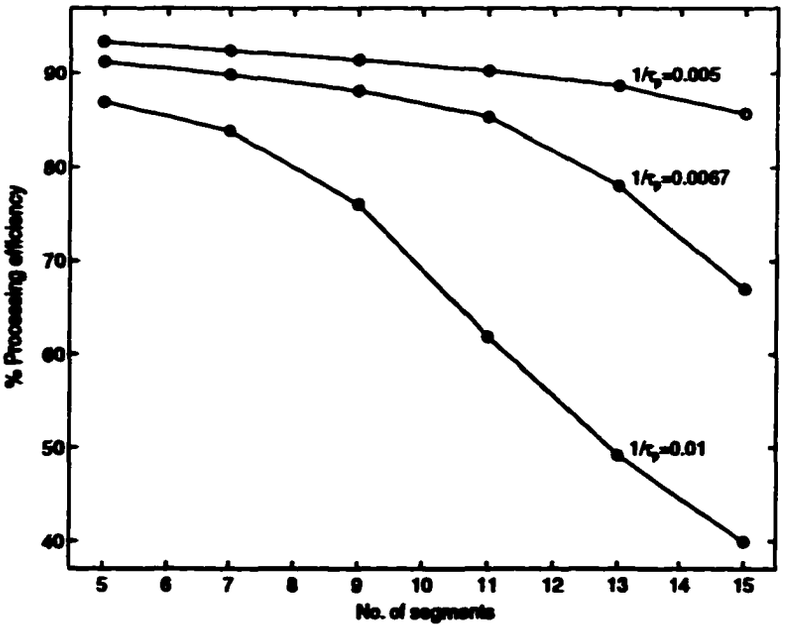


Figure 5.2: Processing efficiency of single flit buffer model under uniform memory reference: $n = 20$, $m = 20$, $b = 10$, $\tau_m = 4$ and $t = 3$.

As we mentioned earlier, the pipelining property of wormhole routing could make the network latency largely insensitive to path length if there is no contention. Adve and Vernon [41] showed that poor performance in the torus network with wormhole routing is mainly caused by the inherent latency of communication rather than by contention in the network. Thus, performance in such low dimensional direct networks is likely to be latency-limited rather than bandwidth-limited when processors block after each request.

In an SMBS with finite flit buffers, however, bus contention tends to contribute to blocking to a greater extent. When contention is an issue, the pipelining property contributes greatly to low system performance, along with the finite size of flit buffers. Hence, the effect of bus contention in the SMBS is expected to be more profound because processors must share segment buses with memory modules as well as with segment switches. The effect of bus contention on performance in bus-based systems could be as significant as distance related latency in direct networks. Hence, a plausible way for reducing the probability of contention is to increase flit buffer size.

To study how performance changes with different flit buffer size, we examined the two extreme cases that set bounds on the performance of systems: namely systems with single flit buffers and systems with infinite flit buffers. We also studied the behavior of systems with packet-sized flit buffers based on event-driven simulations, along with the two extreme cases. Figures 5.3 and 5.4 show respectively, a comparison of the behavior of mean response time and processing efficiency, with different

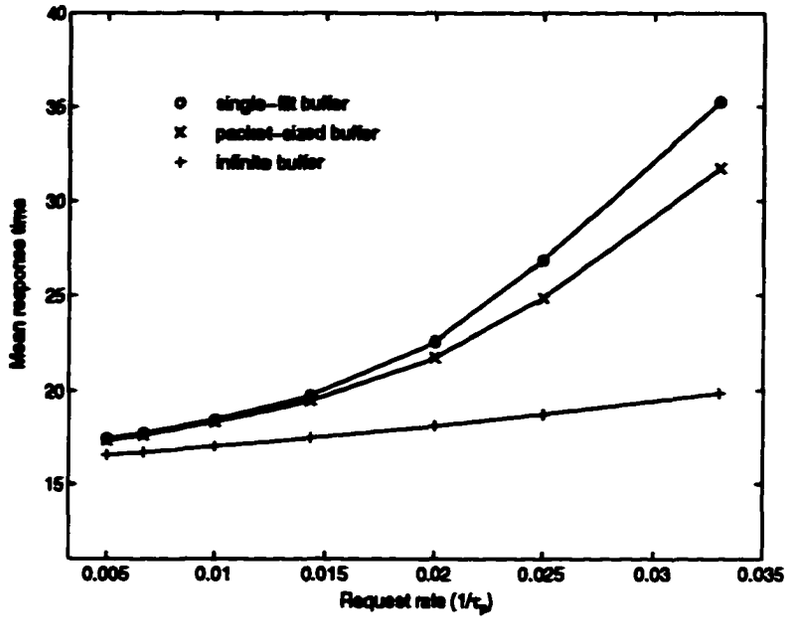


Figure 5.3: Comparison of mean response time with different flit buffer sizes: $n = 10$, $m = 10$, $b = 10$, $g = 9$, $\tau_m = 4$ and $t = 3$.

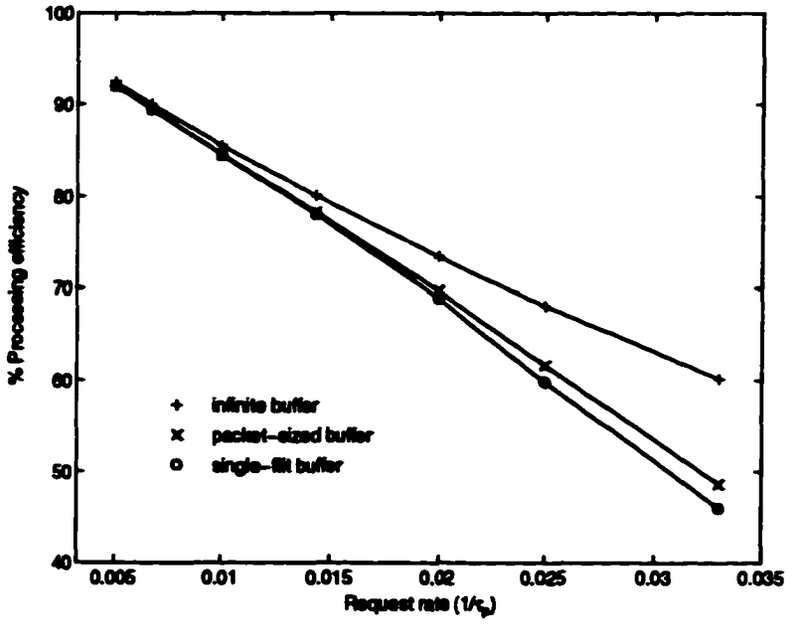


Figure 5.4: Comparison of processing efficiency with different flit buffer sizes: $n = 10$, $m = 10$, $b = 10$, $g = 9$, $\tau_m = 4$ and $t = 3$.

flit buffer sizes. In both figures, there are no significant performance changes among different flit buffer size cases when request rates are low ($1/\tau_p < 0.015$). As the request rate becomes higher, the performance gap between the infinite flit buffer and finite flit buffer cases becomes progressively larger. However, the performance gap between the single flit buffer and packet-sized flit buffer cases is relatively small for different request rates. The performance degradation of the finite flit buffer case as request rate increases is largely attributed to increased blocking caused by the finite buffering spaces. Thus, we conclude that the performance of the SMBS with finite flit buffers deteriorates by contention rather than by network distance. In an SMBS, increasing flit buffer size reduces the probability of contention and provides performance improvement. Furthermore, this property allows the cut-through advantage of wormhole routing to be fully exploited.

We examine how well the SMBS with infinite flit buffers scales. Figure 5.5 shows that the scalability of the infinite flit buffer model under even uniform memory reference is good as processing efficiency decreases slowly with increasing number of segments. For larger segment size in Figure 5.6, we observe that processing efficiency exhibits a slow linear down-slope, even though its overall absolute value is generally lower than that in the case in Figure 5.5 (with smaller segment size). Although performance decrease becomes larger at high request rates ($1/\tau_p \geq 0.02$), the linear down-slope in Figure 5.6 shows clearly that decrease in processing efficiency is mainly due to the increased network distance that a packet must encounter rather than bus contention.

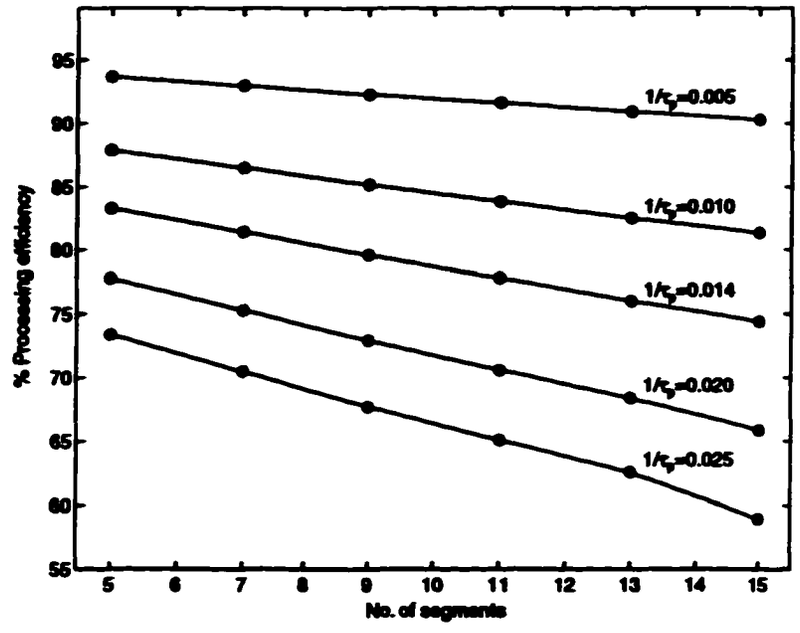


Figure 5.5: Processing efficiency of infinite flit buffer model under uniform memory reference: $n = 10$, $m = 10$, $b = 10$, $\tau_m = 4$ and $t = 3$.

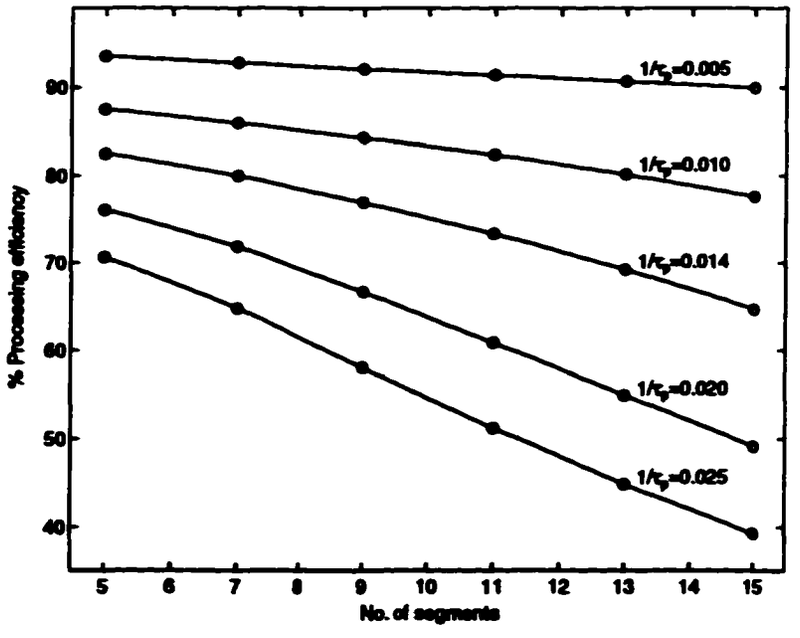


Figure 5.6: Processing efficiency of infinite flit buffer model under uniform memory reference: $n = 20$, $m = 20$, $b = 10$, $\tau_m = 4$ and $t = 3$.

Next, we compare the performance of the two SMBS configurations, a ring connection of segments and a linear connection of segments under a uniform memory reference model. The ring connection offers an average of about 33% reduced network delay over a linear connection. Figures 5.7 and 5.8 depict the change in mean response time and processing efficiency of a linear connection and a ring connection of 9 segments for different request rates.

In Figure 5.7, the difference between the two lines in the infinite flit buffer case indicates the amount of the reduced network delay contributed by the reduced network distance due to the symmetry of the ring connection. Of course, the reduced network delay also includes the effect of balanced utilization of segment buses contributed by the symmetry of the ring connection. We can observe from the figures that increase in contention due to higher request rates is not sufficiently compensated by some reduction in network distance. Figure 5.8 depicts the processing efficiency difference between the two connection topologies under uniform memory reference. Figures 5.7 and 5.8 assure us, once again, that (segment bus) contention in the SMBS is as significant as the effect of network distance latency in direct networks.

The results presented thus far assume a uniform memory reference model. We next examine the impact of memory reference locality on performance. Applications exhibiting memory reference locality can take advantage of architectural features of the SMBS to realize performance gains. An application with high locality is expected to reduce bandwidth demands on the system. This in turn could

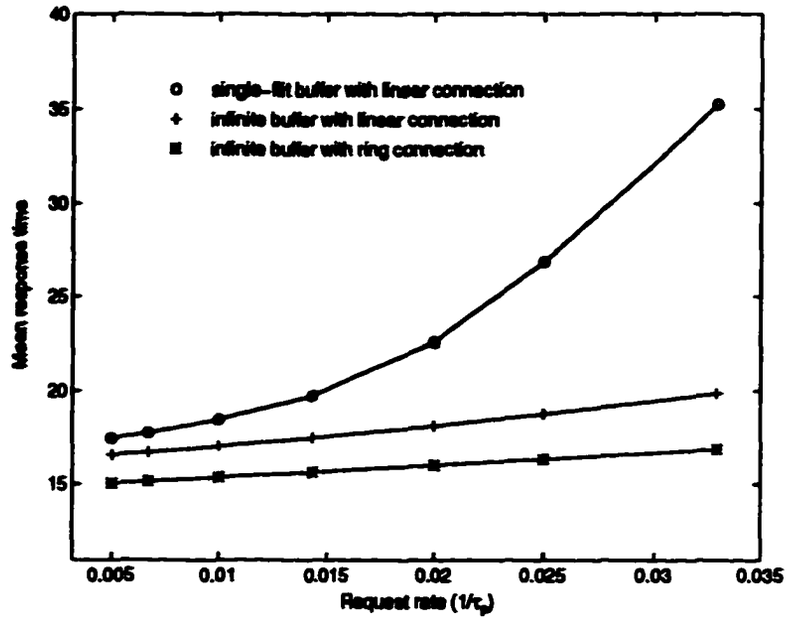


Figure 5.7: Comparison of mean response time with the two different connection topologies (linear and ring connections): $n = 10$, $m = 10$, $b = 10$, $g = 9$, $\tau_m = 4$ and $t = 3$.

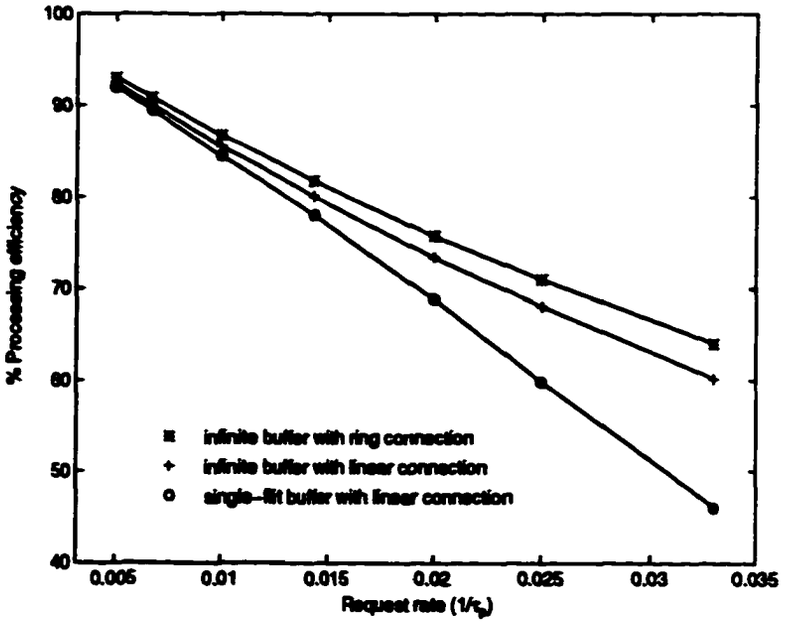


Figure 5.8: Comparison of processing efficiency with the two different connection topologies (linear and ring connections): $n = 10$, $m = 10$, $b = 10$, $g = 9$, $\tau_m = 4$ and $t = 3$.

cause the contention effect to be reduced significantly. As we assumed earlier, for a given locality, a fraction of requests is directed to local memory modules within the originating segment and the remaining requests are evenly distributed to non-local memory modules (outside the originating segment). Figure 5.9 shows processing efficiency of a single flit buffer model as a function of the number of segments for varying memory reference localities. The figure is shown for $n = 10$, $m = 10$, $b = 10$, $1/\tau_p = 0.02$, $\tau_m = 4$ and $t = 3$. Figure 5.10 depicts processing efficiency of a single flit buffer model for $1/\tau_p = 0.01$ and larger segment size: $n = 20$, $m = 20$ and $b = 10$. In both figures, processing efficiency improves substantially with increasing locality because locality reduces bus contention as well as average network distance.

As can be seen in Figures 5.5 and 5.9, the performance of the single flit buffer case with locality of 0.4 is comparable to that of the infinite flit buffer case under uniform memory reference. For larger segment size, as seen in Figures 5.6 and 5.10, the performance of the single flit buffer case with locality of 0.6 is as good as that of the infinite flit buffer case. From the above observations, we conclude that although the scalability of single flit buffer models is generally poor because of high bus contention, the performance of the single flit buffer models with proper locality is significantly improved. For example, the performance of single flit buffer models with locality = 0.4 to 0.6 (depending on segment size) is as good as that of the infinite flit buffer models under a uniform memory reference assumption. As seen earlier, the scalability of the infinite flit buffer models is observed to be good.

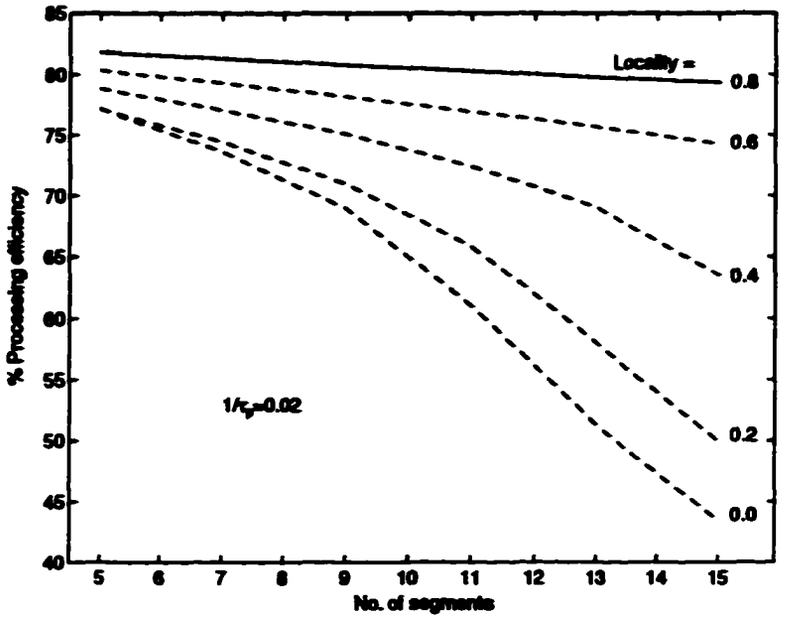


Figure 5.9: Effect of memory reference locality for single flit buffer model: $n = 10$, $m = 10$, $b = 10$, $\tau_m = 4$, $1/\tau_p = 0.02$ and $t = 3$.

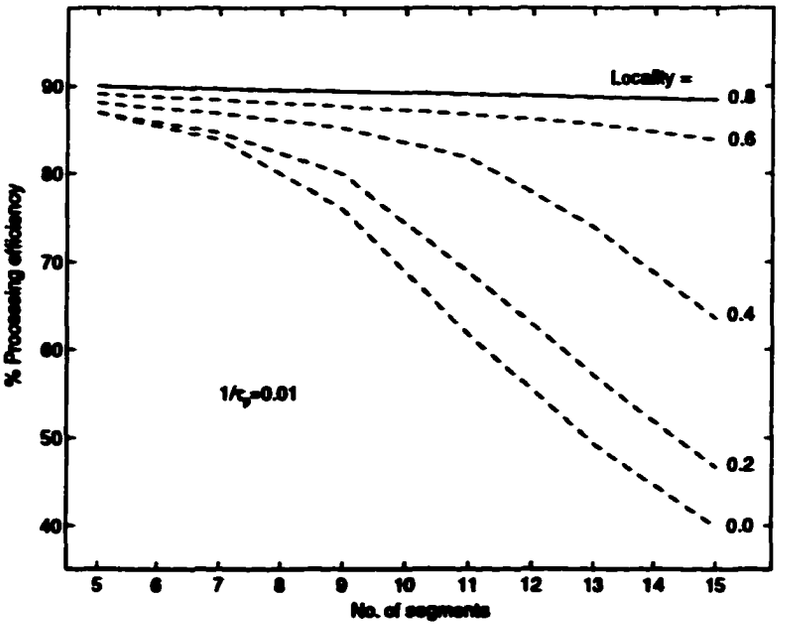


Figure 5.10: Effect of memory reference locality for single flit buffer model: $n = 20$, $m = 20$, $b = 10$, $\tau_m = 4$, $1/\tau_p = 0.01$ and $t = 3$.

Figures 5.11 and 5.12 depict processing efficiency of infinite flit buffer models of the two different segment sizes with varying memory reference localities. In both figures, processing efficiency is improved with increasing locality rates. We notice that increasing memory reference locality improves processing efficiency more rapidly in the case of a single flit buffer model than in the case of an infinite flit buffer model. The positive effect of memory reference locality in the single flit buffer case, on performance, is more profound than that in the infinite flit buffer case because network contention in the single flit buffer case is much higher.

The number of flits per packet, t, is another factor that can affect system performance. In direct networks, when there is no contention and packet length is relatively large, network latency is almost insensitive to network distance. Contrary to direct networks, the SMBS is more susceptible to network contention because it is a bus-based system. Hence, the SMBS favors a small number of flits per packet because the small number of flits reduces the possibility of blocked packets occupying simultaneously several network resources. Figures 5.13 and 5.14 show performance degradation with increasing the number of flits per packet. As request rates become high, performance deterioration is severe as we might expect.

We assumed earlier that a basic segment consists of 10 processors, 10 memory modules and 10 segment buses. Although it is difficult to find any relationship to performance among system configuration parameters n, m, b and g, it is interesting to observe how performance changes with varying system parameters. Here, we observe the effect of the number of segment buses on performance.

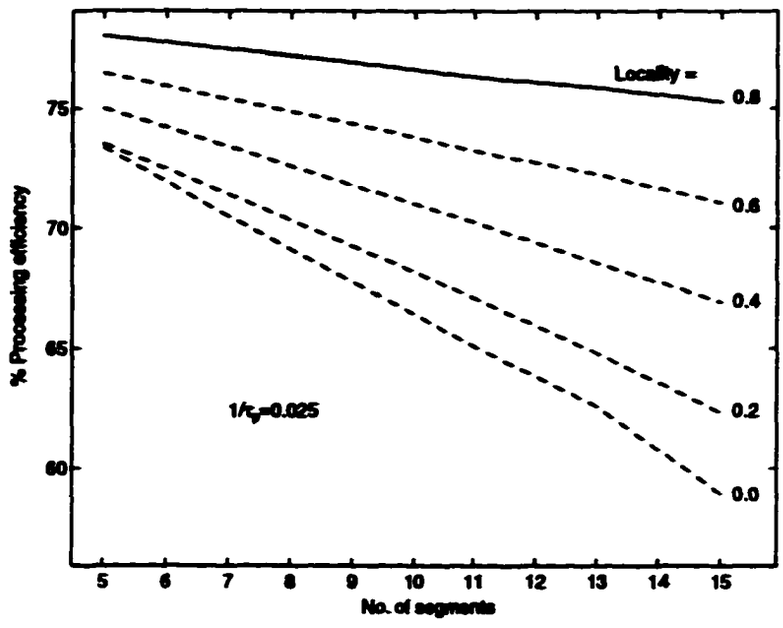


Figure 5.11: Effect of memory reference locality for infinite flit buffer model: $n = 10$, $m = 10$, $b = 10$, $\tau_m = 4$, $1/\tau_p = 0.025$ and $t = 3$.

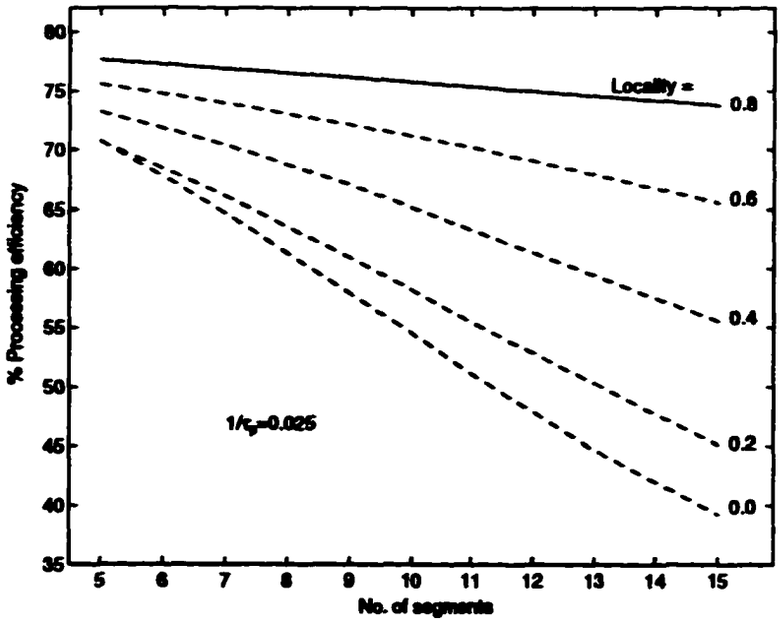


Figure 5.12: Effect of memory reference locality for infinite flit buffer model: $n = 20$, $m = 20$, $b = 10$, $\tau_m = 4$, $1/\tau_p = 0.025$ and $t = 3$.

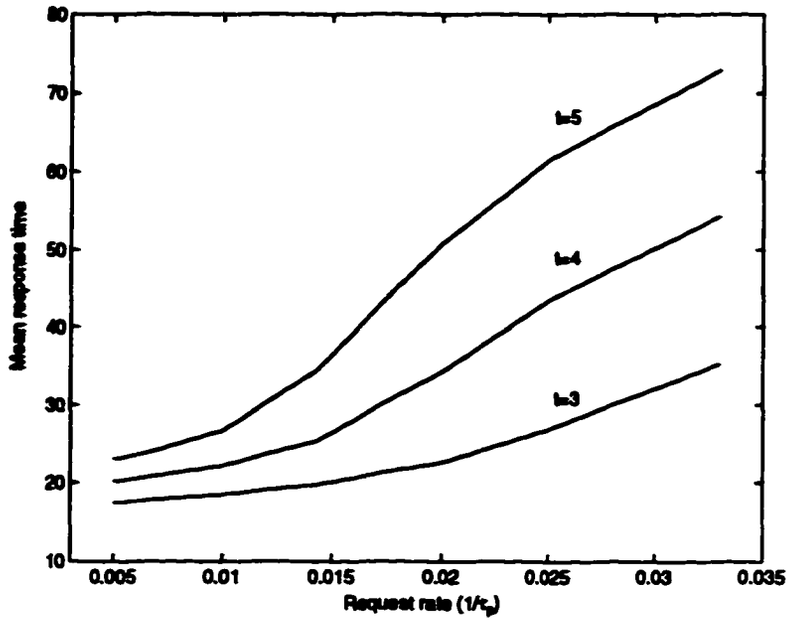


Figure 5.13: Mean response time with varying flit numbers per packet: $n = 10$, $m = 10$, $b = 10$, $g = 9$ and $\tau_m = 4$.

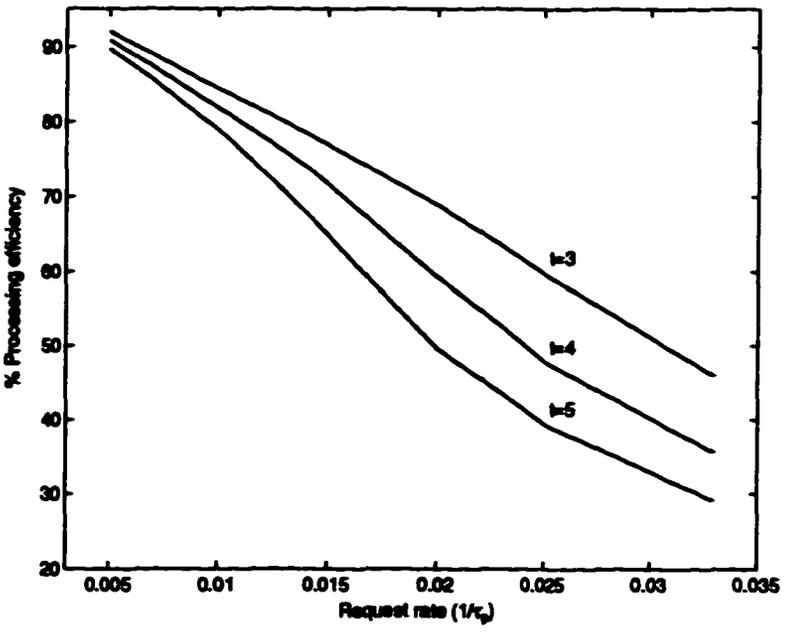


Figure 5.14: Processing efficiency with varying flit numbers per packet: $n = 10$, $m = 10$, $b = 10$, $g = 9$ and $\tau_m = 4$.

Figure 5.15 shows mean response time comparison, for the single flit buffer case, with varying number of segment buses per segment. This is shown for $n = 10$, $m = 10$, $g = 5$, $\tau_m = 4$ and $t = 3$ under a uniform memory reference assumption. In Figure 5.15, we observe that the mean response time increases rapidly when the number of segment buses in a segment is reduced below 8 for the given parameter setting. While it is difficult to predict quantitatively the threshold at which abrupt performance degradation occurs, we can explain it qualitatively. We can explain the abrupt change by stating that shortage of segment buses increases bus contention which in turn increases blocking. Increased blocking further increases contention. This cycle of negative effects stimulates the abrupt performance degradation. Therefore, we can point out, at least, the fact that the performance of an SMBS is very sensitive to the effect of bus contention.

Figure 5.16 shows a comparison of processing efficiency for the single flit buffer case under a uniform memory reference assumption with a varying number of segment buses per segment.

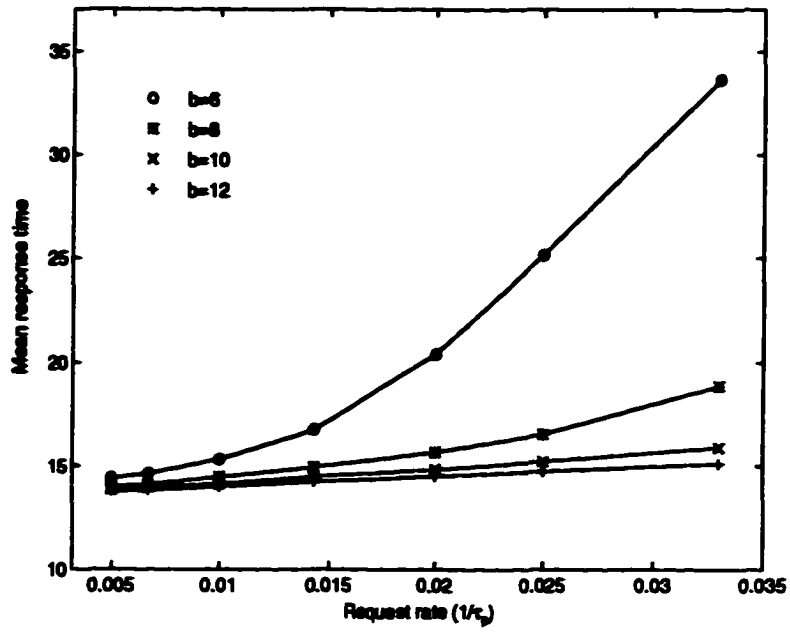


Figure 5.15: Mean response time of single flit buffer case with varying segment buses per segment: $n = 10$, $m = 10$, $g = 5$, $\tau_m = 4$ and $t = 3$.

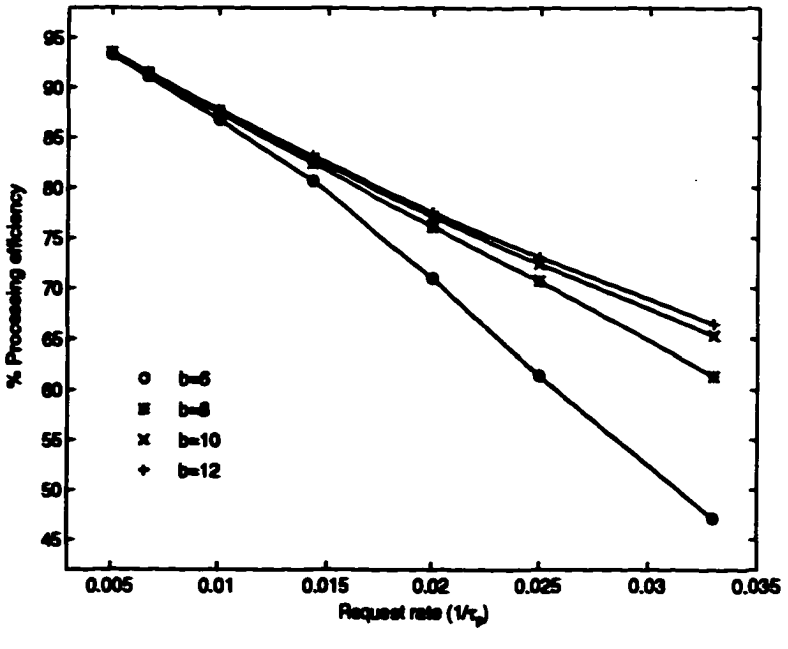


Figure 5.16: Processing efficiency of single flit buffer case with varying segment buses per segment: $n = 10$, $m = 10$, $g = 5$, $\tau_m = 4$ and $t = 3$.

CHAPTER 6

CONCLUSION

We have introduced and investigated a new class of bus-based multiprocessor systems called the Segmented Multiple Bus System (SMBS). The new architecture offers increased scalability while maintaining the bus-based shared memory system's advantages in terms of high degree of fault tolerance, ease of expansion and ease of programming.

One of the unique characteristics of the SMBS is that it is composed of smaller, bus-based, full connection partitions called segments. A set of segment switches positioned between adjacent segments serve the function of electrically isolating (or buffering) the adjacent segments. Thus, in the SMBS, the bus loading problem associated with traditional multiple bus systems is not an issue if segments are not too large, although each segment is a full connection multiple bus system. Another interesting feature of the SMBS is that it supports wormhole switching which is traditionally used in direct network topologies.

Moreover, through segmentation, a very fast bus cycle is possible and the overall bandwidth of the SMBS is increased proportionally with the number of segments. However, increasing the number of segments also increases latency by a corresponding amount. Together, both segmentation and wormhole routing make the system scalable even though the SMBS is a bus-based system.

The segmented system can be classified as a non-uniform memory access network although each segment is a full bus connection subsystem. In such a non-uniform memory access network, applications whose memory reference patterns tend to favor near memory modules can run efficiently. Therefore, the segmented system can utilize memory reference locality which is usually not the case in conventional bus-based systems. By exploiting the locality of memory reference inherent to many applications, scalability of the SMBS could be drastically improved.

One of the most important design aspects of a multiple bus system is the arbitration mechanism. In the SMBS, we have employed a distributed parallel arbitration mechanism. In each segment, a two-phase arbitration scheme has been employed to resolve memory module and segment bus access conflicts. In the first phase, the segment bus is assigned to a segment switch if a request exists at the segment switch. In the second phase, both processor selection and bus assignment are carried out in parallel in order to reduce arbitration and assignment delay. Each segment has an arbitration system that is identical in all other segments and arbitration in each segment is independent of those in other segments. Hence, the arbitration delay in the SMBS (although it is relatively large) is dependent not on system size but on the size of a segment; i.e., arbitration delay is not a function of the number of segments in the system. This is another attractive feature of the SMBS.

The SMBS is a cascaded connection of segments. To scale-up an SMBS with a linear connection, additional segments can be added at the ends. In a ring connection of segments, the ring can be opened and additional segments inserted into

the ring. This simple and modular way of system extension gives the designer more flexibility in terms of system scalability.

We have developed accurate queueing models for wormhole routed SMBS's with single and with infinite flit buffers employed at each segment switches. This, to our knowledge, has been the first attempt to adapt wormhole routing to a bus-based nondirect network. Using approximate MVA, we have analytically evaluated mean residence times at a segment switch, a processor and a memory module. These were utilized to evaluate mean response time of a memory request and processing efficiency which we have used as performance metrics.

Our analytical queueing models are novel and new because they include features of both direct and nondirect networks. We have included the effect of blocking and that of pipelining associated with wormhole routing. The blocking effect has been captured in computing the mean waiting time for a header flit in order to compete for a segment bus at a segment switch. In the calculation of the flit residence time at a segment switch, we have taken into account the flit's movement in relation to the other flits in the same packet by using pipelining property of wormhole routing. The bus group of each segment has been modeled as a flow equivalent service center representing multiple servers with a single central queue. The central server has been considered to be load dependent.

To reduce bus contention, we have assumed that a transient packet has priority over others in the context of bus assignment. The effect of this priority has been

reflected in larger bus access time for non-transient packets than for transient packets.

To check the effect of the assumptions and approximations introduced in the analytical queueing models, we have executed extensive simulations without any approximations. The results obtained from analytical models have been found to be very close to those obtained from simulations. We have investigated how system performance changes with different flit buffer sizes and with different connection topologies. We have addressed how system performance changes by varying the number of segment buses and the number of flits per packet. We have also studied the performance under a uniform memory reference pattern and examined the impact of memory reference locality on the system's scalability. The positive effect of memory reference locality in the single flit buffer case has been found to be more profound than that in the infinite flit buffer case. This is because segment bus contention is inherently higher in the single flit buffer case. We can summarize our conclusions on system scalability as follows.

- Under a uniform memory reference assumption, an SMBS with single flit buffers does not scale well, however, the system with infinite flit buffers shows good scalability.

- Memory reference locality improves system performance for both the single flit buffer and infinite flit buffer cases because locality reduces bus contention as well as average network distance. The improvement in performance of the

single flit buffer case is more substantial than that of the infinite flit buffer case because bus contention is inherently higher in the single flit buffer case.

- The performance of the single flit buffer models with locality in the 0.4 to 0.6 range (depending on segment size) is comparable to that of the infinite flit buffer models under a uniform memory reference assumption (which show good scalability).

We have made several observations which compare our results with the generally known results in the context of wormhole routed low dimensional direct networks. Such observation can be summarized as follows:

- While performance in low dimensional direct networks is generally network distance limited, performance in an SMBS with finite flit buffers is likely to be bus contention limited.
- Increasing in flit buffer size in the SMBS reduces bus contention and results in significant performance improvement.
- If the number of flits per packet is relatively large, the negative effects of long network distance in low dimensional direct networks are minimized. Contrary to direct networks, the SMBS is likely to be adversely affected by an increased number of flits per packet since it would lead to high levels of bus contention.
- A small number of flits per packet and a wide bus width will help to reduce the probability of contention and consequently improve the performance of the SMBS.

There are still several issues open for future research. We summarize these as follows.

- We have developed and evaluated system performances for two extreme cases: the single flit buffer case and the infinite flit buffer case. It would be worthwhile to develop an analytical performance model for the SMBS with any finite-sized flit buffers. Although this is a difficult problem, solving this problem would help in determining the flit buffer size required to match the performance of an infinite flit buffer case to within a certain percentage.

- The performance of an SMBS was shown to be very sensitive to the effect of bus contention. Thus, it is important to study the impact and role of caches on reducing bus traffic, and consequently on bus contention. Research on performance improvement in SMBS's caused by introducing caches would complement our research.

- The approach we adopted in developing the analytical models is rather comprehensive in the sense that the models incorporate features of both direct and nondirect networks. This approach can possibly be applied to several other network topologies which are classified as direct and/or nondirect networks. For instance, if the shared memory in an SMBS is distributed to the processors (becomes local to processors), the model would be easily applicable to the resulting message passing environment.

- Developing a deadlock-free routing algorithm for a ring connection of segments with finite flit buffers would be interesting work. In a direct network, deadlock

can be generally avoided using one of two approaches: dimension ordering [6, 11, 12] and virtual channels [31]. However, the fact that a segment bus is a resource shared by many and that it is arbitrated among many requesters might require a new way of thinking.

BIBLIOGRAPHY

[1] T. Y. Feng, "A Survey of Interconnection Networks," *IEEE Comput. Mag.*, Vol.14, No.12, December 1981, pp. 12–27.

[2] W. A. Wulf and C. G. Bell, "C.mmp - A Multi-Mini-Processor," *Proc. Fall Joint Comput. Conf.*, AFIPS, December 1972, pp. 756–777.

[3] T. Lang, M. Valero, and I. Alegre,"Bandwidth of Crossbar and Multiple-Bus Connections for Multiprocessors," *IEEE Trans. Comput.*, Vol. C-31, No.12, December 1982, pp. 1227–1234.

[4] T. N. Nudge, J. P. Hayes, and D. C. Winsor, "Multiple Bus Architectures," *IEEE Comput.*, June 1987, pp. 42–48.

[5] H. J. Siegel, *Interconnection Networks for Large Scale Parallel Processing: Theory and Case Studies*, Lexington Books, Lexington, MA, 1984.

[6] H. Sullivab and T. R. Bashkow, "A Large Scale, Homogeneous, Fully Distributed Parallel Machine," *ACM Proc. 4th Annual Int. Symp. Comput. Architecture, IEEE*, March 1977, pp. 105–124.

[7] J. T. Kuehn and B. J. Smith, "The HORIZON Supercomputing System: Architecture and Software," *Proc. Supercomputing '88*, November 1988.

[8] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, J. Hennessy, M. Horowitz, and M. Lam, "Design of the Stanford DASH Multiprocessor," Comput. Syst. Lab. TR 89-403, Stanford Univ., December 1975.

[9] A. Agarwal, B. H. Lim, D. A. Kranz, and J. Kubiatowicz, "APRIL: A Processor Architecture for Multiprocessing," *ACM Proc. 17th Annual Int. Symp. Comput. Architecture*, June 1990, pp. 104–114.

[10] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Distributed Comput.*, Vol.1, No.3, 1986, pp. 187–196.

[11] Intel Corporation, *A Touchstone DELTA System Description*, 1991.

[12] C. L. Seitz, W. C. Athas, C. M. Flaig, A. J. Martin, J. Seizovic, C. S. Steele, and W. K. Su, "The Architecture and Programming of the Ametek Series 2010 Multicomputer," *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications*, Pasadena, CA, January 1988, pp. 33–36.

[13] M. D. Noakes, D. A. Wallach and W. J. Dally, "The J-Machine Multicomputers: An Architectural Evaluation," *20th Annual Int. Symp. Comput. Architecture*, May 1993, pp. 224–235.

[14] W. C. Athas and C. L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Comput. Mag.*, Vol.21, August 1988, pp. 9–24.

[15] H. Jiang and K. C. Smith, "PPMB: A Partial Multiple Bus Multiprocessor Architecture for Improved Cost Effectiveness," *IEEE Trans. Comput.*, Vol.41, No.3, March 1992, pp. 361–366.

[16] E. D. Lazowska, J. Zahorja, G. S. Grahm, and K. C. Sevcik, *Quantitative System Performance - Computer System Analysis using Queueing Network Models*, Englewood Cliffs, NJ: Prentice-Hall, 1984.

[17] K. L. Johnson, "The Impact of Communication Locality on Large-Scale Multiprocessor Performance," *ACM Proc. 19th Annual Int. Symp. Comput. Architecture*, May 1992, pp. 392–402.

[18] T. Lang, M. Valero, and M. A. Fiol, "Reduction of Connections for Multibus Organization," *IEEE Trans. Comput.*, Vol. C-32, No.8, August 1983, pp. 707–715.

[19] W. T. Chen and J. P. Sheu, "Performance Analysis of Multiple Bus Interconnection Networks with Hierarchical Requesting Model," *IEEE Trans. Comput.*, Vol.40, No.7, July 1991, pp. 834–842.

[20] S. M. Mahmud, "Performance Analysis of Multilevel Bus Networks for Hierarchical Multiprocessors," *IEEE Trans. Comput.*, Vol.43, No.7, July 1994, pp. 789–805.

[21] M. N. Karim, "Design and Analysis of Reduced Connection Multiple Bus Systems: A Probabilistic Approach," Ph.D.'s Dissertation, Electrical and Computer Engineering Dept., Louisiana State University, December 1996.

[22] A. Ghafoor, A. L. Goel, J. K. Chan, T. M. Chen, and S. Sheikh, "Reliability Analysis of a Fault Tolerant Multi-Bus Multiprocessor System," *Proc. 3rd IEEE Symp. Parallel and Distributed Processing*, Dallas, TX, December 1991, pp. 436–443.

[23] A. Varma, "Combinatorial Design of Bus-Based Interconnection Structures," *Research Report RC 12550 (#54752)*, IBM, Yorktown Heights, New York, September 1986, pp. 309–312.

[24] E. Lugue, D. Rexachs, J. Sorribes and A. Ripoll, "A Modular Arbitration System for Multiple Buses Multiprocessors," *Microcomputers, usage and design*, Euromicro 1985, pp. 579–585.

[25] R. C. Pearce, J. A. Field, and W. D. Little, "Asynchronous Arbiter Module," *IEEE Trans. Comput.*, Vol. C-22, No.9, September 1975, pp. 931–932.

[26] W. W. Plummer, "Asynchronous Arbiters," *IEEE Trans. Comput.*, Vol. C-21, No.1, January 1972, pp. 37–42.

[27] P. Corsini, "n-User Asynchronous Arbiter," *Electronics Letters*, Vol.11, No.1, January 1975, pp. 1–2.

[28] L. Morin and H. F. Li, "Design of Synchronisers: A Review," *IEE Proc.* Vol.136, Pt.E, No.6, November 1989, pp. 557–564.

[29] T. Lang and M. Valero, "M-Users B-Servers Arbiter for Multiple Buses Multiprocessors," *Microprocessing and Microprogramming*, Vol.10, 1982, pp. 11–18.

[30] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Comput. Network*, Vol.3, May 1987, pp. 547–553.

[31] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Comput.*, Vol. C-36, No.5, May 1989, pp. 547–553.

[32] W. J. Dally and P. Song, "Design of a Self-Timed VLSI Multicomputer Communication Controller," *Proc. Int'l Conf. Computer Design, IEEE*, Los Alamitos, CA, 1987, pp. 230–234.

[33] M. A. Marsan and M. Gerla, "Markov Models for Multiple Bus Multiprocessor Systems," *IEEE Trans. Comput.*, Vol. C-31, No.3, March 1982, pp. 239–248.

[34] K. B. Irani and I. Onyuksel, "A Closed-Form Solution for the Performance Analysis of Multiple-Bus Multiprocessor Systems," *IEEE Trans. Comput.*, Vol. C-32, No.11, November 1984, pp. 1004–1012.

[35] D. Towsley,"Approximate Models of Multiple Bus Multiprocessor Systems," *IEEE Trans. Comput.*, Vol. C-35, No.3, March 1986, pp. 220–227.

[36] Q. Yang and S. G. Zaky, "Communication Performance in Multiple-Bus Systems," *IEEE Trans. Comput.*, Vol.37, No.7, July 1988, pp. 848–853.

[37] Q. Yang and L. N. Bhuyan, "Analysis of Packet-Switched Multiple-Bus Multiprocessor Systems," *IEEE Trans. Comput.*, Vol.40, No.3, March 1991, pp. 352–357.

[38] W. J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Comput.*, Vol.39, No.6 June 1990, pp. 775–785.

[39] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Trans. Parallel and Distributed Systems*, Vol.2, No.4, October 1991, pp. 398–412.

[40] S. L. Scott and J. R. Goodman, "The Impact of Pipelined Channels on k-ary n-cube Networks," *IEEE Trans. Parallel and Distributed Systems*, Vol.5, No.1, January 1994, pp. 2–16.

[41] V. S. Adve and M. K. Vernon, "Performance Analysis of Mesh Interconnection Networks with Deterministic Routing," *IEEE Trans. Parallel and Distributed Systems*, Vol.5, No.3, March 1994, pp. 225–246.

[42] J. Kim and C. R. Das, "Hypercube Communication Delay with Wormhole Routing," *IEEE Trans. Comput.*, Vol.43, No.7, July 1994, pp. 806–814.

[43] K. G. Shin and S. W. Daniel, "Analysis and Implementation of Hybrid Switching," *IEEE Trans. Comput.*, Vol.45, No.6, June 1996, pp. 684–692.

[44] J. R. Jackson, "Jobshop-Like Queueing Systems," *Management Science*, Vol.10, No.1, October 1963, pp. 131–142.

[45] D. L. Willick and D. L. Eager, "An Analytical Model of Multistage Interconnection Networks," *Proc. ACM SIGMETRICS Conf. on Measurement Modeling Comput. Syst.*, May 1990, pp. 192–202.

[46] D. Chaiken, C. Fields, K. Kurihara, and A. Agarwal, "Directory-Based Cache-coherence in Large-Scale Multiprocessors," *IEEE Comput. Mag.*, Vol.23, June 1990, pp. 41–58.

[47] P. A. Lewis, A. S. Goodman, and J. M. Miller, "A Psuedo-Random Number Generator for the System/360," *IBM Systems Journal*, Vol.8, No.2, 1969, pp. 136–146.

[48] H. Kobayashi, *An Introduction to System Performance Evaluation Methodology*, Addison-Wesley, 1978.

APPENDIX: TERMINOLOGY

$R[s]$ Mean response time of a class s request.

$R_{proc}[s]$ Mean residence time that the header flit of a class s packet experiences at the head of a processor output queue.

$R_{network}[s]$ Weighted sum of mean residence times of a class s packet in the forward path and in the return path.

$R_{sd,network}$ Mean residence time in the network of a class s request packet from the first segment switch in the forward path to the target memory module in segment d.

$R_{ds,network}$ Mean residence time in the network of a class s reply packet from the first segment switch in the return path to the requesting processor.

$R_{mem}[s]$ Mean residence time of a class s packet in a memory module.

$R_{d|s,mem_{in}}$ Sum of the mean delay that a class s request packet experiences in the input queue of a target memory module in segment d and the memory service time, τ_m.

$R_{d|s,mem_{out}}$ Mean delay that a class s reply packet experiences in the output queue of a target memory module in segment d.

$\rho_{proc}[s]$ Mean processor utilization of class s.

$r_{i,sd}(1)$ Mean residence time of the header flit of a class s packet at a segment switch in segment i in the forward path from s to d.

$r_{i,ds}(1)$ Mean residence time of the header flit of a class s packet at a segment switch in segment i in the return path from d to s.

$r_{i,sd}(k)$ Mean residence time of the k^{th} flit of a class s packet at a segment switch in segment i in the forward path from s to d.

$r_{i,ds}(k)$ Mean residence time of the k^{th} flit of a class s packet at a segment switch in segment i in the return path from d to s.

$r_i(k)$ Mean residence time of the k^{th} flit at a segment switch in segment i.

$u_i(k)$ Mean utilization of a segment switch in segment i by the k^{th} flit.

$w_i(1)$ Mean waiting time for a segment bus in segment i by a header flit.

$u_{i,p_{out}}(k)$ Mean utilization of a processor output queue by the k^{th} flit of a class i packet.

$u_{i,m_{out}}(k)$ Mean utilization of a memory output queue in segment i by the k^{th} flit.

$r_{i,p_{out}}(k)$ Residence time of the k^{th} flit of a class i packet at a processor output queue.

$p_{proc|mem}$ Probability that a processor participates in memory arbitration in the case of an intra-segment access.

$p_{sw_{up}|mem}$ Probability that a segment switch in an upper bus group participates in memory arbitration.

$p_{sw_{lo}|mem}$ Probability that a segment switch in a lower bus group participates in memory arbitration.

$p_{proc|bus}$ Probability that a processor for an inter-segment access participates in bus arbitration.

$p_{mem|bus}$ Probability that a memory module participates in bus arbitration.

$p_{p_{succ}|mem}$ Probability that a request from a processor succeeds in memory arbitration in the case of an intra-segment access.

$p_{proc|bus_{eff}}$ Probability that a processor successfully participates in bus arbitration.

$s_{d|s,p_{out}}$ Mean waiting time of the header flit of a class s packet directed d in order to compete for a segment bus at a processor output queue in segment s.

$\gamma_{d|s,p_{out}}$ Mean residual residence time of a tail flit in service that the header flit of a class s packet observes when it arrives at the first segment switch in the path from s to d.

$t_{s,bus_{up}}$ Mean upper segment bus access time at segment s.

$t_{s,bus_{lo}}$ Mean lower segment bus access time at segment s.

$t_{s,bus_{up}access}$ Expected time interval between consecutive bus service completions in the upper bus group of segment s.

$t_{s,bus_{lo}access}$ Expected time interval between consecutive bus service completions in the lower bus group of segment s.

$w_{d|s,m_{in}}$ Mean time that a class s request packet experiences in the input queue of a target memory module in segment d.

$s_{d|s,m_{out}}$ Delay experienced by the header flit of a class s packet at the head of a memory output queue in segment d until a segment bus is allocated to it.

$q_{d|s,m_{in}}$ Mean memory input queue length that is observed by a class s request packet when it arrived at a memory input queue in segment d.

$q_{d|s,m_{out}}$ Mean memory output queue length that is observed by a class s reply packet when it arrived at a memory output queue in segment d.

$u_{d|s,m}$ Probability that a memory module is busy serving a request when a class s packet arrived at the memory module in segment d.

$w_{d|s,m_{out}}$ Mean queueing delay until the header flit of a class s reply packet arrives at the head of a memory output queue in segment d.

$r_{d|s,m_{out}}(1)$ Residence time of the header flit of a class s reply packet at the head of a memory output queue in segment d.

$\gamma_{d|s,m_{out}}$ Mean residual residence time of the tail flit in service that a class s reply packet observes when the header flit arrived at the first segment switch in the return path from d to s.

$\mathcal{K}_{s,p_{out}}$ Integer random variable that represents the number of occupied queue heads observed by a class s request packet when it arrived at the head of a processor output queue.

$T[\mathcal{K}_{s,p_{out}}]$ Mean time until a class s packet packet can get access to a segment bus while $\mathcal{K}_{s,p_{out}}$ other queue heads are occupied.

$\eta_{s,p}$ Number of occupied processor output queue heads observed by a request that has just arrived at the head of a processor output queue in segment s.

$\eta_{s,m}$ Number of occupied memory output queue heads observed by a request that has just arrived at the head of a processor output queue in segment s.

$\mathcal{K}_{d,m_{out}}$ Integer random variable that represents the number of occupied queue heads observed by a reply packet, when it arrived at the head of a memory output queue in segment d.

$T[\mathcal{K}_{d,m_{out}}]$ Mean time until a reply packet can get access to a segment bus in segment d while $\mathcal{K}_{d,m_{out}}$ other queue heads are occupied.

$\theta_{d,p}$ Number of occupied processor output queue heads observed by a reply packet when it arrives at the head of the memory output queue in segment d.

$\theta_{d,m}$ Number of occupied memory output queue heads observed by a reply packet when it arrives at the head of a memory output queue in segment d.

γ_m Mean residual lifetime of a memory request in service at a memory module.

$u_{i,bus}$ Probability that a segment bus in segment i is busy serving a transient packet at the beginning of the second arbitration phase.

$q_{i,sw}$ Mean queue length seen by an arriving header flit at an infinite flit buffer queue in segment i.

$p_{bus_i,busy}$ Probability that a header flit observes segment bus i busy serving a request from a local processor or a local memory module.

Φ_s Number of requests that participate in bus assignment of the second arbitration phase at segment s.

$\phi_{s,p}$ Number of processors participating in bus assignment of the second arbitration phase at segment s.

$\phi_{s,m}$ Number of memory modules participating in bus assignment of the second arbitration phase at segment s.

VITA

Jungjoon Kim was born in Taegu, Korea. He received the Bachelor of Science degree in electrical engineering from Kyungpook National University, Korea, in 1981. After he got the Master of Science degree in electrical engineering from Korea Advanced Institute of Science and Technology in 1983, he has worked as an instructor at Kyungpook National University from 1983 to 1984. Since November 1984, he has joined Korea Telecom Research Laboratories, where he has worked in the Electronic Switching System Division. He is currently on leave of study to pursue the Doctor of Philosophy degree at Louisiana State University.

He is married to Yoonkyung Lee. They have two sons, Steven Seokwon and Justin Seokhyun.

DOCTORAL EXAMINATION AND DISSERTATION REPORT

Candidate: Jungjoon Kim

Major Field: Electrical Engineering

Title of Dissertation: Performance and Analysis of Segmented Multiple Bus Systems

Approved:

Major Professor and Chairman

Dean of the Graduate School

EXAMINING COMMITTEE:

Date of Examination:

June 18, 1997