**ON THE PARALLEL IMPLEMENTATION OF OSI PROTOCOL
PROCESSING SYSTEMS**

By

Robert Lam

B. A. Sc., The University of British Columbia, 1989

**A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE**

in

**THE FACULTY OF GRADUATE STUDIES
DEPARTMENT OF ELECTRICAL ENGINEERING**

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

September 1993

© Robert Lam, 1993

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

(Signature)

Department of Electrical Engineering
The University of British Columbia
Vancouver, Canada

Date October 8, 1993.

Abstract

In a heterogeneous computing environment, computers have to use a suitable transfer syntax to communicate with each other because of the differences in data representations in each computer. The need to translate the data from one format to another, which can be understood by all computers, takes over 90% of the processing power used for protocol processings. Application specific architectures in a heterogeneous system may not be efficient in performing protocol processing functions and there is a need for a stand alone protocol processing system. In a gigabit networking environment, the processing power required for protocol processing is beyond that of a single processor and a multiprocessor approach is needed. Three different multiprocessor architectures with increasing complexities for protocol processing are described in this thesis. These designs make use of the properties of locality of processing and prefetching techniques. Both simulations and analytical methods are used to evaluate the performance. The results indicated that by putting instructions and packet data close to the processors, a significant increase in processing throughput can be obtained as compared to previously published designs. In our best scenario, we obtained a throughput of up to 560MBits/s. With further optimizations, it is expected that the throughput can be extended to a gigabit rate. The study also revealed that a bus based architecture is sufficient in handling the OSI protocol processing requirement for a network rate of over 560MBits/s. It was observed that the use of fast processors alone without matching fast memory is inadequate in providing a feasible solution. The sequential nature of protocol definition could be changed to include more parallelism and to further improve the scalability of the protocol processing system.

Table of Contents

Abstract	ii
Table of Contents	iii
List of Tables	vi
List of Figures	viii
Acknowledgements	x
1 Introduction	1
1.1 Guide to the rest of the thesis	2
1.2 Background on protocol processing	2
1.3 Motivation for the research into a multiprocessor design	6
1.4 Background on parallel processing	7
1.5 Related Research Activities	7
1.6 Goals and approach of study	9
2 Design issues	11
2.1 System requirement	11
2.2 Method of parallelization	11
2.3 Single queue versus multiple queues	13
2.4 Slow versus fast processors	15
2.5 Abstract Model of the system	19
2.5.1 Interconnection requirement	21
2.5.2 Number and kind of buses needed	21

2.5.3	Locking mechanism	23
2.5.4	Analysis	24
2.5.5	Other parameters for measuring the performance	30
2.5.6	Other major components	31
2.6	The simple shared memory design	34
3	The local instruction memory design	41
3.1	Background on locality	41
3.2	Effects of locality	42
3.3	Performance analysis and Simulations	43
3.3.1	Merits of the LIM design	49
3.3.2	Weakness of the LIM design	50
4	The packet relocation design	51
4.1	Usefulness of data cache	51
4.2	Spatial locality achieved through copying	52
4.3	Performance analysis and Simulation	55
4.3.1	Merits of the SPR design	60
4.3.2	Weakness of the SPR design	60
4.4	Overcoming the weakness	64
4.4.1	Reduce the number of packet relocation before processing starts . . .	64
4.4.2	Reduce the number of packet relocation during processing	65
4.4.3	Discussions	73
5	Results and Discussions	76
5.1	Cost	76
5.2	Memory speed	77
5.3	Bus speed	77
5.4	Multiple connections	78

5.5	Packet lengths	78
5.6	Other presentation layer operations	80
5.7	Future Work	80
6	Conclusions	81
	Bibliography	84
A	Simulation Details	87
A.1	Simulation model	87
A.2	Protocol processing data	89
A.3	Simulation parameters	91
A.3.1	Obtaining Peak Performance	91
A.3.2	Selecting the content of the workload	91
A.4	Analysis of accuracy	93
A.4.1	Random numbers	93
A.4.2	Sensitivity to random numbers	93
A.4.3	Picking the simulated time	95
B	queueing model	97
B.1	Multiple queue multiple server	97
B.2	Single queue multiple server	97
B.3	Comparing single queue and multiple queue systems	98
C	Simulation Results	101
D	Glossary of terms	106

List of Tables

1.1	OSI 7 layers of protocols	3
1.2	Processing power required for protocol processing in a 1000 megabit network	6
2.1	Memory access type	26
2.2	Summary of memory access based on static code count	28
2.3	Hit ratios for simulating the SSM design	36
2.4	Parameters used for simulating the SSM design	36
3.1	Hit ratios for simulating the LIM design	44
3.2	Variation of maximum throughput with processor speed	49
4.1	Hit ratios for simulating the SPR design	55
4.2	Variations of maximum throughput when the burst access speed is changed .	61
4.3	Average processing time using 10mips processors in a 0.5s interval	62
4.4	Average processing time using 20mips processors in a 0.5s interval	63
4.5	Average processing time using 40mips processors in a 0.5s interval	63
4.6	Average processing time for the IPR design in a 0.5s interval	69
4.7	Parameters used for finding the max throughput	70
4.8	Hit ratios for finding the max throughput	70
4.9	System performance when the input rate is higher than the processing throughput	73
5.1	Summary of system characteristics	76
A.1	Number of access to PDUs for the different layers	89
A.2	Number of state access for the different layers	89

A.3	Number of instruction access for the different layers	90
A.4	Effects on Throughput with different workload	92
A.5	Standard Deviation of Peak Throughput(MBytes/s)	94
A.6	Standard Deviation of Random Access Bus Usage	94
A.7	Standard Deviation of Burst Access Bus Usage	95
A.8	Variation due to different simulated time	96
C.1	SSM design with 10mips processors	101
C.2	SSM design with 20mips processors	101
C.3	SSM design with 40mips processors	102
C.4	LIM design with 10mips processors	102
C.5	LIM design with 20mips processors	102
C.6	LIM design with 40mips processors	103
C.7	SPR design with 10mips processors	103
C.8	SPR design with 20mips processors	103
C.9	SPR design with 40mips processors	104
C.10	IPR design using 40mips processors	104
C.11	IPR design using 40mips processors and 2 times transfer syntax operations .	104
C.12	IPR using 40mips processors and fast memory	105

List of Figures

1.1	OSI layer concept	4
2.1	A single queue multiple server system	13
2.2	a multiple queue multiple server system	14
2.3	Variation of effective processor cycle time for different instruction hit ratio . .	17
2.4	Variation of effective processor cycle time for different data hit ratio	17
2.5	Variation of effective processor cycle time for different ratio of register only to all instructions	18
2.6	Abstract model of the protocol processing system	20
2.7	Determining the average number of retries needed for getting a lock	24
2.8	Shared resources in the system	25
2.9	Block diagram of the SSM design	34
2.10	Number of times a packet is moved during processing	35
2.11	Analytical and Simulation results of the throughput for the simple shared mem- ory design	37
2.12	Bus utilizations of the random access and burst access bus for the SSM design	39
2.13	Throughput and bus utilization of the SSM design with 20mips and 40mips processors	40
3.1	The Local Instruction Memory design	44
3.2	Analytical and Simulation results of the throughput for the LIM design . . .	46
3.3	Bus utilizations of the random access and burst access bus for the LIM design	47
3.4	Throughput and bus utilization of the LIM design with 20mips and 40mips processors	48

4.1	The Simple Packet Relocation design	53
4.2	Number of packet relocations for the SPR design	54
4.3	Analytical and Simulation results of the throughput for the SPR design . . .	57
4.4	Bus utilizations of the random access and burst access bus for the SPR design	58
4.5	Throughput and bus utilization of the SPR design with 20mips and 40mips processors	59
4.6	Comparison of the LIM design and the SPR design with different burst access speeds	62
4.7	Number of packet relocations for the IPR design	67
4.8	Throughput of the IPR design using 40mips processors	68
4.9	Resource utilization of the IPR design using 40mips processors	69
4.10	Throughput of the IPR design using 40mips processors and fast memory . . .	71
4.11	Resource utilization of the IPR design using 40mips processors and fast memory	72
5.1	Maximum throughput variation with different packet lengths	79
A.1	Overview of the simulation program	88
A.2	Throughput variation with contents of workload	92
B.1	A multiple queue multiple servers system	98
B.2	A single queue multiple servers system	99

Acknowledgements

I would like to express my sincere thanks to Dr. Mabo Ito for his guidance throughout the course of my research. I gratefully acknowledge the valuable comments, suggestions and simulation tools provided by Leonard Takeuchi. I would also like to thank my committee, in particular Dr. Hussein Alnuweiri, for their insightful suggestions that enhanced the quality of my thesis.

I would like to extend my appreciation to Murray Goldberg and Mark McCutcheon from the high-speed networking group at UBC for answering my questions on protocol processing.

A very special thanks to IBM Canada Laboratory for granting me the education leave of absence so that I can continue my education. Special thanks also go to the Natural Sciences and Engineering Research Council of Canada for awarding me the post graduate scholarship.

Thanks also go to Barry Tsuji, William Low and Jeffrey Chow for the enlightening discussions throughout the course of my work. Lastly, I would like to thank my mom, my dad and my sisters for their support, encouragement and understanding without which the work presented in this thesis would not be possible.

Chapter 1

Introduction

High performance computer architectures have been designed to deal with different computational needs. However, a single application will usually consist of a number of tasks having different computational requirements. One single architecture is thus unable to satisfy the needs of every application and peak performance of the architecture can not be achieved as the system must spend part of its time working on computations not designed for it. A heterogeneous computing environment solves this problem by having a number of application specific architectures in the system so that each portion of the task can be processed in the fastest possible rate [13].

Computers within a heterogeneous computing environment have to communicate with each other to exchange information. Since each computer may have its own data format, a suitable transfer syntax is needed to enable the exchange of data. The OSI protocol stack provides a model which computers can use to communicate. In a high performance computer system, a high speed network would be required to handle the communication traffic. The need to perform transfer syntax conversion in a high performance computer system turns out to be a major burden computationally because a large amount of data has to be transformed in a short period of time. The problem becomes worse with the use of application specific architectures in the heterogeneous environment because these architectures may not be efficient in performing protocol processing operations. Thus, a stand alone protocol processing system is desirable.

An example of such a system may consist of a database server machine optimized for data access, a vector processor for computations and a workstation for user interface. Protocol

processing systems capable of high speed operations could be installed at the database server and the vector processor as these specialized machines would not be optimized for protocol processing. The workstation may or may not be equipped with an external protocol processing system depending on the traffic and the cost of such a system.

1.1 Guide to the rest of the thesis

The remaining sections in this chapter describe the OSI protocol reference model and some basics of parallel processing. Related research activities, motivations for the research, our approach and objectives are also discussed. Chapter 2 explains the major design issues that have to be made for the multiprocessor protocol processing system. An analytical model is developed for a simple shared memory design and the behavior of this design is examined using both the analytical approach and the simulation approach. Chapter 3 deals with the effects of locality and studies the performance improvements that is obtained with the improved architecture. Chapter 4 discusses other means of achieving locality and thus increasing the system throughput by means of a further improved architecture. Chapter 5 gives a summary of the results and discusses their implications. Chapter 6 draws some conclusions based on the results. Details about the simulation program, data and simulation results are given in the appendices.

1.2 Background on protocol processing

Open System Interconnection (OSI) is a reference model defined by the International Organization for Standardization (ISO) for the communication between computer systems [32]. The OSI model consists of 7 layers, each of which has a unique function as defined by a service definition and a protocol specification. The service definition specifies the activity between adjacent layers and the protocol specification defines the interaction between peer entities on the communicating machines. The seven layers include the application layer, the presentation layer, the session layer, the transport layer, the network layer, the datalink layer and the

physical layer. A brief description of each layer is given in table 1.1.

Layer	Description
Application	deals with the distributed information services and access of the OSI environment by users.
Presentation	deals with the data representation and includes conversions, compression and encryption of data.
Session	deals with the establishment, maintenance and termination of connections between co-operating applications.
Transport	deals with the reliable and transparent transfer of data between end points. Segmentation and reassembly of packets are also performed if the network packet size is fixed and smaller than the session packet size.
Network	deals with the establishment, maintenance and termination of connections. In a connection oriented system, routing table has to be maintained to determine the path for routing packets.
Data link	deals with the reliable transfer of information across the physical link.
Physical	deals with the mechanical, electrical, functional and procedural characteristics to access the physical medium.

Table 1.1: OSI 7 layers of protocols

Communication between two systems is achieved by passing information in the form of protocol data units (PDUs) between peer layer entities. Since all peer layers are only connected logically except the physical layer, PDUs have to move down the layers until they get to the physical layer and then move up to the peer layer on the destination machine. Information is passed between adjacent layers using service data units (SDUs) through service access points (SAP). Figure 1.1 shows the interactions between the PDUs and the layer entities. An (N+1) layer entity trying to communicate with its peer entity with a (N+1) PDU will send the PDU as a (N) SDU to the (N) layer entity in its own machine by invoking a (N) service primitive through a (N) SAP. The (N) layer entity will then communicate with its peer with a (N) PDU created by adding protocol control information (PCI) to the (N) SDU¹. This will continue until the physical layer is reached and at which point, the information is sent.

¹The protocol control information is commonly known as the header.

Figure 1.1: OSI layer concept

On receiving a (N) PDU, the (N) layer entity will carry out the actions specified in the PCI. The (N) SDU will also be passed to the (N+1) layer entity if required.

Protocol processing for connection-oriented protocols can be divided into two parts, packet processing and connection control block manipulation. Packet processing includes the transfer syntax conversion, checksum calculation, encryption and decryption, and encoding and decoding of headers. Connection control block manipulation [15] includes state machine transitions, enqueueing of received packets and other synchronization operations. Connection control block manipulation requires only a few lines of code for each packet while packet processing is more complicated and could take 20 or more instructions per byte of packet data.

In our design, we are assuming that the host machine is responsible for the application layer processing and the network adapter is responsible for the physical layer processing. Therefore, our protocol processing system is only responsible for the presentation layer to the datalink layer. In our discussion, processing associated with sending a PDU is referred to as downward processing as it involves moving the PDU down the protocol stack. Processing associated with receiving a PDU is referred to as upward processing for a similar reason.

Upward processing is more complicated and time consuming than downward processing. This is because for upward processing, we have to reassemble packets into bigger ones in the transport layer and since packets could come in out of sequence, special care must be taken to make sure that all the required packets have arrived before carrying on with the processing. Decoding of the packets in upward processing also requires more time than encoding in downward processing. For decoding, we do not know what is coming in and have to decode the data bit by bit whereas for encoding, we have all the information required. The need to compute the checksum for error detection also makes upward processing very time consuming. For downward processing, checksum calculation can be done in parallel with the data copy operations and thus no extra time is required. The definition of OSI uses a finite

state machine and thus the actual sequence of the PDUs has to be preserved. In a uniprocessor system, this could be achieved easily but for a multiprocessor system, PDUs can get out of sequence and explicit means are needed [15] to ensure that they are kept in order. This requirement of processing packets in sequence also makes upward processing more difficult than downward processing.

1.3 Motivation for the research into a multiprocessor design

One of the major difficulties in OSI protocol processing at a gigabit network rate is the large amount of processing power required for the transfer syntax conversion operations which account for more than 90% [9] of all the protocol processing power. The computational power needed can not be easily met with a single processor if we would like to make use of all the communication bandwidth available to us. Table 1.2 gives the processing power required for different transfer syntax conversion complexities in terms of the number of instructions per byte ². For a gigabit network, data is coming in at a rate of 125MBytes/s and the processing power needed is tabulated as shown in table 1.2. Even with an Alpha processor,

number of instructions/byte	processing power (mips)
10	1250
20	2500
40	5000

Table 1.2: Processing power required for protocol processing in a 1000 megabit network

which could provide up to 400mips of processing power, we see that in a gigabit network, a number of them would still be needed. Since we are building the protocol processing system as a peripheral device to the host, we would like to keep the cost down by using slower processors. Therefore, in terms of cost and technology, a multiprocessor approach provides a possible solution. The use of a multiprocessor design could also allow an incremental increase in processing throughput if not all of the available network bandwidth is needed now and also

²refer to appendix A for details

provides a higher fault tolerancy than a single processor system.

1.4 Background on parallel processing

The speed at which a processor runs at depends a lot on the technology used in fabricating the processor. For some applications, the processing power required exceeds that which can be offered by a single processor. A practical solution in those situations is to incorporate a number of processors into the system instead of having only one. A multiprocessor system could either speed up the processing of a task by dividing the task into subtasks and working on these subtasks in parallel or increase the system throughput by pipelining tasks to raise the rate of task completion. The former approach would benefit applications which can be subdivided into smaller tasks easily while the latter approach is more suitable for applications where new tasks can be started in a shorter time than the actual processing time of each task. The performance improvement will be affected by the granularity of the subtasks or pipeline stages.

Many types of multiprocessor architectures have been proposed and Flynn [12] has classified them based on the number of instruction and data streams. In a single instruction multiple data streams (SIMD) machine, the same operation is executed on each datum in a uniformly structured data set. In a multiple instructions multiple data streams (MIMD) system, there are several independent computers each capable of executing its own program and consequently the resulting system is more suited to less structured operations.

The choice of a particular architecture would be very dependent on the characteristics of the application itself. More discussions on the system architecture is given in the next chapter.

1.5 Related Research Activities

Research work on protocol processing techniques for high speed networks has increased with the advent in optical network technology. Different approaches have been taken by researchers

to tackle the high processing power requirement for OSI protocol processing. One approach is to speed up the transfer syntax conversion operations as these are the most CPU cycle consuming operations of protocol processing. Huitema et al. [18] proposed a replacement of the ASN.1 basic encoding rules with a light weight syntax to improve the performance. They found that an improvement of 6 times was possible and a higher increase could be obtained for some particular data structures. The problem with using a new transfer syntax is that it will take a long time before it can be accepted as a standard. Joseph [23] parallelized the transfer syntax conversion operations using specialized RISC processors. He was also able to speed up the processing by about 5 times. The limitation of Joseph's approach is that the speedup depends on the complexity of the data packet and so for simple data types, the speedup is low.

Another approach is to deal with the lower protocol layers only in the protocol processing system and leave the higher layers to the host. Kanakia et al. [25] designed a highly specialized network adapter board (NAB) for the light weight VMTP transport protocol using a multiprocessor system. Giarizzo et al. [14] exploited the inherent parallelism in protocols by using a multiprocessor-based communication subsystem. Braun et al. [3] described a transputer based approach for implementing a gateway. The results of these studies proved that a multiprocessor approach could handle high speed networks with a throughput of more than 100Mbits/s at the transport layer. Jain et al. [21] increased the throughput further to Gbits/s rates with their parallel processing design. The problem of leaving the higher layers to the host is that with a high network rate, the host will have to spend most of its time doing protocol processing and not on the application using those data. Out of the large number of research efforts, only a few of them have attempted to deal with the higher protocol layers. Goldberg et al. [15] and Zitterbart [40] described ways of parallel processing the protocol stack up to and including the presentation layer using a small number of processors and showed that a significant improvement in throughput can be obtained.

Takeuchi [37] presented a couple of designs which could provide higher layer processings using up to 50 processors with a processing throughput of 100MBits/s. The first design included both local and shared memory. In that design, the shared memory was used to store the state information and other synchronization variables while the local memory was used for everything else, including packets. Packets coming into the system were assigned to the processors using a round robin scheme. The second design used only shared memory. All instructions, packet data and state information were stored in the global shared memory. The result of Takeuchi's research indicated that the interconnection between the processors and also the job allocation algorithm significantly affects the processor utilization, which in turn results in the nonlinear increase in system throughput with the number of processors. The shared memory design was found to scale poorly and was not able to provide a 100MBits/s throughput because of bus contentions. The mixed memory design was able to give a throughput of 100MBits/s but suffered from a low processor utilization.

1.6 Goals and approach of study

Our research attempts to solve the problems observed in Takeuchi's design and to look at higher speed networks. Designs presented in this thesis are based on the principal of locality and software cache prefetching techniques. In software cache prefetching, hit ratios are improved by predicting what data will be used next and loading them into the cache before they are used. In our case, we would like to use a similar idea to reduce the amount of shared memory contentions. It is believed that with the specific knowledge of the application, we could put the data in the best possible location at any given time to ensure a high degree of temporal and spatial locality. An incremental approach is taken here to look for a suitable architecture. Designs with increasing complexities are studied. Both analytical and simulation methods are used to evaluate our results. Objectives of the research are summarized as follows:

- Design a system that could handle the OSI processing requirement of a gigabit network.

- Design such a system in a simple and economical way. This is important because we do not want the protocol processing unit, which is designed as a peripheral, to be more complicated or expensive than the host machine it is attached to.
- Design a system with good linear increase in throughput and high utilization of all system resources.

Chapter 2

Design issues

In the design of a multiprocessor system, many design issues have to be solved. The following sections describe some of the system requirements and the major choices that have been made to optimize the system performance. An analytical method for determining the system performance is also discussed.

2.1 System requirement

In this research, we would like to design a multiprocessor based protocol processing system capable of all the processing required from layer 2 through layer 6 in the OSI protocol stack. The design should maximize the resource utilizations and processing throughput at a reasonable complexity and cost.

2.2 Method of parallelization

Protocol processing can be parallelized in a number of ways based on the granularity of the unit of parallelization [15][40]. The atomic unit can be a connection, a protocol layer, a function or a packet. The resulting multiprocessor system would either increase the peak throughput or speed up the processing as other parallel processing systems discussed in section 1.4.

For a processor per connection system, improvement of performance is obtained from the concurrent processing of different connections. The processing time remains the same and the increase in system throughput depends on the number of concurrent connections and the amount of activity in each. If the load is not balanced for each of the connections, the

effectiveness of a parallel design is low as only those heavily used connections can contribute to the improvement in performance.

Using protocol layers as a unit of parallelization would again increase the system throughput. The system works as a pipeline and its performance is governed by the slowest stage. The high processing power needed for the presentation layer and the synchronization needed between layers makes this design undesirable. We could improve this design by subdividing the presentation layer into more pipeline stages but we would also introduce more synchronization overhead which may offset any improvement.

In the processor per function design, the processing time is reduced from the concurrent processing of different protocol functions. The performance is highly dependent on the amount of embedded parallelizable tasks in the protocol. A high synchronization overhead would be incurred because of this sort of parallelism.

The final approach uses a processor per packet design. This design improves the system throughput without speeding up the processing of individual packets. With this arrangement, each packet is assigned to a processor which would carry out all processings required for that packet. Since the processor will carry the packet through all the layers, synchronization between layers as in the processor per layer design is not required. Without regard of which connection a packet belongs, the system can still derive the parallelism out of the processing as long as there are packets in the system. Inherent parallelism needed for processor per function design is not required. The only drawback of the processor per packet approach is that it is possible for some packets to pass each other in the stack and extra synchronization operations are needed to maintain the logical order of the packets. The proper order can be ensured by the use of internal sequence number [15] which adds to the overhead. Despite the extra overhead, this design is taken because of the higher potential gain in processing throughput.

With the packet per processor design, we are parallelizing the transfer syntax conversion operations which normally do not require access to the shared state information and do

not need any synchronization for the encoding and decoding functions. Multiple processors could be performing transfer syntax conversions for a number of packets in parallel once their headers which must be processed sequentially have been worked on. Each processor can be executing different instructions at any time because different packets could be in different layers and a MIMD type of architecture would be suitable for this application.

2.3 Single queue versus multiple queues

In a per packet based model, each packet forms a basic unit for parallel processing. A decision arises as to whether these units of work should be put into a single queue and distributed to the processors when the latter are ready or there should be a queue for each processor and these basic work units put into these queues according to some scheduling algorithm. These two configurations are shown in figures 2.1 and 2.2.

Figure 2.1: A single queue multiple server system

These two approaches can be compared by using the classical queueing theory. The delay time per packet of the single queue design can be proved to be less than that of the multiple queue design. The detailed analysis is given in appendix B. Intuitively, it could

Figure 2.2: a multiple queue multiple server system

be seen that a single queue design will give the maximum processor utilization as none of the processors in such a system can become idle if there is something in the queue. On the contrary, for a multiple queue system, processors may be idle even while some queues are not empty depending on how intelligent the scheduling algorithm is in estimating the processing time for each packet. The sequential nature of protocol processing in which packets have to be processed in a predefined sequence could complicate the scheduling algorithm for the multiple queue system and therefore worsen the situation. Some of the processors may also be forced to become idle even if their job queues are non empty if they do not have the packet with the proper sequence number. Takeuchi [37] found that processor utilization of the multiple queue system could be below 50%.

The single queue structure has been chosen in our designs because of its better delay time characteristics, higher resource utilization and its simplicity in terms of implementation. The multiple queue system is not considered further in our discussion.

2.4 Slow versus fast processors

In protocol processing, packet data have to be accessed a number of times for checksum calculation, transfer syntax conversion and reassembly operations [2]. There is very little immediate reuse of these packet data after they are accessed each time and thus normal caching techniques are not effective in increasing the system speed. The use of caches for this kind of operations may be harmful as useful data may be displaced from the cache [35] when the packet data are accessed. Without the benefits of using a cache, packet data accesses would have to go through the main memory most of the time. The access rate would be dependent on the memory speed. Since there is a gap between processor speed and memory cycle time [16], no real benefits can be obtained by using a fast processor if the main memory speed can not be increased with it.

The effective speed of a processor given the memory cycle time can be estimated given the knowledge of the application statistics. To develop an analytic model, the following notations are used:

x	=	portion of register only instructions out of all instructions
h_i	=	instruction hit ratio
h_d	=	data hit ratio
t_c	=	cache cycle time
t_p	=	processor execution time (for reg only instructions)
t_m	=	memory cycle time
t_{ave}	=	average system execution time
t_i	=	average instruction execution time

Assuming that if an instruction is in the cache, there is no extra overhead in accessing the instruction and that an instruction fetch starts when an instruction is being executed.

Thus

$$t_i = [h_i * 0 + (1 - h_i)(t_m - t_p)] + t_p$$

$$\text{Let } t_m = \beta t_p \text{ and } t_c = t_p$$

where β is the ratio of memory to processor speed

$$\begin{aligned} t_{ave} &= x t_i + (1 - x)[t_i + h_d t_c + (1 - h_d)t_m] \\ &= t_i + (1 - x)[h_d t_c + (1 - h_d)t_m] \\ &= (1 - h_i)(t_m - t_p) + t_p + (1 - x)[h_d t_c + (1 - h_d)t_m] \end{aligned}$$

by substituting t_i into the equation

$$= (1 - h_i)(\beta - 1)t_p + t_p + (1 - x)[h_d t_p + (1 - h_d)\beta t_p]$$

by substituting t_m into the equation

$$\begin{aligned} &= t_p \{ \beta - 1 - \beta h_i + h_i + 1 + (1 - x)[h_d + (1 - h_d)\beta] \} \\ &= t_p \{ \beta - \beta h_i + h_i + (1 - x)[h_d + \beta - h_d \beta] \} \\ &= t_p \{ \beta - \beta h_i + h_i + (1 - x)\beta(1 - h_d) + (1 - x)h_d \} \\ &= t_p \{ \beta[1 - h_i + (1 - x)(1 - h_d)] + h_i + (1 - x)h_d \} \end{aligned}$$

$$\text{therefore } \frac{t_{ave}}{t_p} = \beta k_1 + k_2$$

where k_1 and k_2 are application specific constants

To achieve peak performance of the processor, $\frac{t_{ave}}{t_p}$ should be as close to 1 as possible. To minimize the effect of β , it is also desirable to have k_1 as close to 0 as possible. In other words, we would like to have the hit rates and percentage of register only instructions to approach 1.

Graphs of $\frac{t_{ave}}{t_p}$ versus β have been plotted for values of h_i , x and h_d as depicted in figure 2.3, figure 2.4 and figure 2.5. The effects of each one of them on the effective processor cycle time is studied by fixing two of the three variables in each of the graphs. We can observe that the effective processor cycle time is most sensitive to the instruction hit ratio and least

Figure 2.3: Variation of effective processor cycle time for different instruction hit ratio

Figure 2.4: Variation of effective processor cycle time for different data hit ratio

Figure 2.5: Variation of effective processor cycle time for different ratio of register only to all instructions

to the percentage of register only instructions. This is because in our application, a high number of instructions is being executed in the syntax conversion phase. The slope of the lines vary greatly for the former case but not for the latter. To ensure that we can utilize at least 50% of the processing power, we would like to keep $\frac{t_{ave}}{t_p}$ to be less than or equal to 2. This value of $\frac{t_{ave}}{t_p}$ can be used to estimate the value of β and the corresponding hit ratios and percentage of register only instructions required.

From figures 2.4, 2.3 and 2.5, we can see that even with high instruction and data hit ratios and a moderately high percentage of register only instructions, we can still only use memories that are around 10 times slower than that of the processor to provide an effective cycle time less than or equal to two times that of the processor. Memory which is slower than the processor by more than 10 times is definitely undesirable as shown in figures 2.4 and 2.5. When the hit ratios are lowered, the speed of the memory must be increased to compensate for the loss.

Keeping a high instruction hit rate is possible by using various prefetching techniques [22][24][26][28][6] but keeping a high data hit rate is a major challenge. Since the data we are referring to here are the packets that keep coming through the system and do not stay in the system for a long time, prefetching them into the cache may not be worthwhile. The cost of issuing prefetches could be high for data packets which vary in length and processing time.

Based on the above discussion, it can be concluded that by using fast processors without matching memory devices, we will not be able to fully utilize the fast processors. The resulting system performance may even be poorer than a slow processor system with matching memory devices. On the other hand, using a high number of slow processors introduces the problem of bus and memory contentions. Thus a tradeoff has to be made in picking the processor speed.

Designs presented in the thesis are restricted to processors with speeds 2 to 8 times faster than memory because of this reason. For a typical memory cycle time of 200ns, the corresponding processor speed is around 10 to 40mips.

2.5 Abstract Model of the system

In the previous sections we have described some of the major issues in the architecture. From our discussions, we have decided that the protocol processing system would be a MIMD system having packets as the unit of parallelization. A single active queue would be used for all jobs to ensure a high resource utilization and low latency. Processor speeds between 10mips to 40mips with the corresponding memory speed at around 200ns for random access will be used.

A simplified model of the system is given in figure 2.6. All packets coming into the system, from the host or the network are enqueued into a single active job queue in a FIFO manner. When a processor from the pool is ready, the first job in the queue is sent to it for processing. Owing to the sequential processing requirement, some jobs cannot be processed

Figure 2.6: Abstract model of the protocol processing system

further when jobs ahead of it have not been finished yet and a strict processor per packet design cannot be realized. These jobs have to be suspended and put into a suspended queue. Since each connection operates independently from others, there are separated suspended queues for each connection. The queues are further divided into upward and downward suspended queues for each connection oriented layer within a single connection. The upward queues are for packets going up the protocol stack and the downward queues are for packets going down the protocol stack.

Jobs in suspended queues will be moved to the active job queue once the preceding job has been processed. When a processor finishes processing a packet in any connection oriented layer, it would look at the suspended queue of that particular connection layer concerned and move the first suspended job to the active queue if it can be processed now.

This simplified model does not take into account of the other resources like the shared memory or the interconnection structure between them. A detailed description of these resources is given in the following sections.

2.5.1 Interconnection requirement

State information indicating the locations of jobs, connection statistics and other synchronization data are used by processors to determine how a job is to be processed. In a multiprocessor system, information can be shared by using a message passing or shared memory paradigm. For the message passing paradigm, the processor sending the message must know which processor will process the packet of the same connection that the sender is working on. This information is not readily available because we do not know which processor is going to process the next packet in that particular connection and thus the sender would have to send the shared data to a well known message server instead. Processors requesting state information have to communicate with this server using a client server model. In such a system, the message server could become a bottleneck.

For a shared memory design, information is written directly into some shared memory locations without going through a server. The particular memory location would become a bottleneck and some explicit locking mechanisms are required to handle concurrent requests. Since the implementation of the server is more complicated than a shared memory design and that the two paradigms would both have a similar contention problem, the simpler shared memory design is employed in our system.

A bus is used to implement the shared memory paradigm because it is the easiest design to implement. By giving each processor the same priority for bus arbitration and using a first come first serve arbitration scheme, we can maintain a fair access to the bus by all processors. Other interconnection networks would be more complicated and should only be considered if the bus design cannot handle the memory traffic.

2.5.2 Number and kind of buses needed

The performance of the shared memory design is affected by the contention problem of the shared resources. Here, we have to deal with both the bus and memory contention problems. Bus contentions occur when more than one processing unit try to acquire the bus and can be

solved by using a bus sufficient system. In a bus sufficient system with m memory modules and p processors, the number of buses $b \geq \min(p, m)$ and there will not be any bus contention. Memory contentions occur when more than one processing unit tries to access the same memory port at the same time and thus can be solved by using multiple memory modules and putting data into different modules to avoid conflicts. These two problems are interrelated and are considered together when we are determining the proper system configuration.

Two major types of access modes can be identified for the protocol processing application. They are the burst access where a whole block of data is being moved and the random access where only a small chunk of data is accessed each time. The burst access mode is used when packets are copied from the interface to the main memory for processing and from the main memory to the interface when processing is done. It is also used for checksum computation when the packet is sent through some specialized hardware¹. Random access is used for the reading and writing of state tables and queues and also for accessing instructions and packet data.

Naturally, it makes sense to have two types of buses to match the access modes. One bus should be optimized for random and another optimized for burst access. Another advantage of this configuration is that dual ported memory devices optimized for this type of operations exists in the form of video random access memory [30].

As mentioned above, the problem of bus contention can be solved by using a bus sufficient system for both buses. However, we could not easily make use of the buses available to us because of memory contentions. Owing to the sequential nature of protocol processing and that the actual processing time for each packet is different, we can not put the packets into the memory modules in such a way that no memory contention will occur without some extensive data movements. The high cost of a bus sufficient system and the time and resource consuming data movements make a bus sufficient system unattractive for our application. Therefore, we are not going to consider a bus sufficient system unless the result reveals that

¹refer to section 2.5.6

there is a need to use more than two buses.

2.5.3 Locking mechanism

In our design, requesters for a particular lock are responsible for getting and releasing the lock.² The requester uses a test-and-set atomic instruction to obtain a lock. If the lock is not available, a busy wait mechanism is used. The requester will wait a certain interval and then retry the test-and-set until the lock is obtained. The choice of the retry interval has a major effect of the system performance. If it is too long, the latency increases. If it is too short, many retries may be needed for getting a lock. These retries will generate extra traffic on the random access bus and affect the speedup behavior.

This simple locking mechanism is used because of its ease of implementation. Since each processor attempts to get a lock independently without any particular co-ordination, it is possible that some processors may never get the lock they want. The severity of this problem will be looked at through simulations.

To determine the number of attempts needed to obtain a lock, the expected value of retry has to be found. This expected value depends on the probability at which a lock is needed and the probability that the lock is locked. The latter would in turn be a function of the system load. Since the system load changes dynamically with the type of packets and also the creation and removal of connections, the exact analysis of the number of attempts is complex and nontractable. Therefore, instead of getting an exact value, a rough estimate of the number of retries is used. To do so, we have assumed a case in which all n processors attempt to get the same lock at the same instant and that once the lock is obtained, the processor concerned will move onto some other jobs and not request the same lock again before all remaining requesters have finished with that lock. Without loss of generality, assume that the processors are numbered in the order in which the lock is acquired as in figure 2.7. Then the last processor has to wait $(n-1)$ time intervals while the first processor

²note that the problem of fault recovery is not discussed here for it is out of the scope of the current work.

does not have to wait. Assume that the lock is held for y units of time and that if a lock is not available, the processor will try again α time units later. In the worst case where the

Figure 2.7: Determining the average number of retries needed for getting a lock

lock becomes available immediately after the processor makes a request, the processor would need to wait for another α time units. Thus the time interval between successful lock requests would be $y + \alpha$ time units³. Using this assumption, we could estimate that the average number of retries is

$$r = \frac{(n-1)(y+\alpha)}{2\alpha} \quad (2.1)$$

2.5.4 Analysis

The performance of a dual bus shared memory system can be estimated by looking at the processing throughput and the usage of the three shared resources, namely the processors, the

³locking is a major problem in parallel systems and more work needs to be done in this area.

random access bus and the burst access bus. The memory is not considered here because we only have a single access path to the memory devices in our design. Therefore looking at the buses would be sufficient because once the bus is obtained, the memory device should also be available. An analytic model and a simulation program are developed to estimate the system processing throughput and resource utilizations. The SSM design proposed by Takeuchi [37] is analyzed using these methods and the results of which are used as a basis for comparison with our designs described in later chapters.

Figure 2.8: Shared resources in the system

Maximum throughput based on the number of processors

Since over 90% of the protocol processing time is spent on the transfer syntax conversion phase [9], we can estimate the maximum throughput based on the processing power needed for transfer syntax conversion.

Let γ = instructions required per byte for transfer syntax conversion

n = number of processors

$$\begin{aligned}
\phi &= \text{effective mips of processor (including the effect of memory speed)} \\
\rho_{processor} &= \text{maximum throughput that could be provided by } n \text{ processors} \\
\text{then } \rho_{processor} &= \frac{n * \phi}{\gamma} \tag{2.2}
\end{aligned}$$

The simple relation shows that throughput can be increased by using more processors which is the main idea of using a multiprocessor system. However, as the number of processors increases, more contentions appear and bus saturation would limit the maximum number that can be used. To get a more accurate estimation, we have to look at the utilization of the shared buses.

Maximum throughput based on the random access bus usage

Assume that we have a Von Neumann architecture. Memory accesses on the random access bus can then be classified as instruction and data accesses. Instruction access refers to the fetching of protocol processing code and data access refers to the reading and writing of packet header, data and other state information. Since we are looking at a complete peripheral, the operating system code is included in the protocol processing code. To facilitate the analysis, instruction access is further divided into transfer syntax conversion related and header processing related accesses. Data access is also divided into state information, packet header and packet data accesses. The following notation is used in our analysis:

Category	Access Type
1	Instruction: Transfer syntax conversion
2	Instruction: Header processing
3	Data: State, Locks
4	Data: Packet header
5	Data: Packet data

Table 2.1: Memory access type

Let γ = average number of instruction access per byte for processing a packet

μ_m = memory cycle time for random access

μ_e = processor execution time

t_i = number of instructions executed per packet for memory access in category i

p_i = fraction of memory access in category i out of all memory access

h_i = hit ratio for category i

l = average length of a packet

ρ = processing throughput

ρ_{random} = maximum processing throughput derived from random access bus usage

p_n = network packet size

r_t, r_s, r_i = # of retries for the transport, session and presentation locks respectively

$$\text{Therefore average hit ratio} = \sum_{i=1}^5 p_i h_i$$

$$\text{and average miss ratio} = 1 - \sum_{i=1}^5 p_i h_i$$

$$\text{average \# of memory access /packet} = (1 - \sum_{i=1}^5 p_i h_i) \sum_{i=1}^5 t_i$$

$$\text{average \# of memory access /s} = \frac{\rho}{l} (1 - \sum_{i=1}^5 p_i h_i) \sum_{i=1}^5 t_i$$

Assume that when a processor needs to access memory, it has to acquire the bus, access the memory location, finish its operation and then relinquish the bus. Thus it will take $\mu_m + \mu_e$ unit of time for each memory access or

$$\text{max \# of memory access/s} = \frac{1}{\mu_m + \mu_e}$$

$$\text{Therefore, } \frac{1}{\mu_m + \mu_e} = \frac{\rho_{random}}{l} (1 - \sum_{i=1}^5 p_i h_i) \sum_{i=1}^5 t_i$$

$$\begin{aligned} \rho_{random} &= \frac{l}{(\mu_m + \mu_e) (1 - \sum_{i=1}^5 p_i h_i) \sum_{i=1}^5 t_i} \\ &= \frac{l}{(\mu_m + \mu_e) (1 - \sum_{i=1}^5 \frac{t_i h_i}{\sum_{i=1}^5 t_i}) \sum_{i=1}^5 t_i} \end{aligned}$$

$$\begin{aligned}
&= \frac{l}{(\mu_m + \mu_e)(\sum_{i=1}^5 t_i - \sum_{i=1}^5 t_i h_i)} \\
&= \frac{l}{(\mu_m + \mu_e)(\sum_{i=1}^5 t_i (1 - h_i))} \text{ MBytes/s} \quad (2.3)
\end{aligned}$$

Based on a static code count of the protocol processing code, t_i and p_i could be determined. The value of h_i depends on the system design and can only be estimated. The average length l depends on the actual traffic and again has to be estimated. Some typical values of t_i have been tabulated in table 2.2. The way in which those values are obtained is discussed in appendix A.

i	access type	number of memory access (t_i)
1	Transfer syntax	γl
2	Header processing	$557.5(\frac{l}{p_n}) + 795$
3	State, Locks	$\frac{l}{p_n}(128 + r_t) + 112 + r_s + r_t$
4	Packet header	$84(\frac{l}{p_n}) + 90$
5	Packet data	$\frac{l}{2}$

Table 2.2: Summary of memory access based on static code count

Since our estimate for the number of retries is a pessimistic value under most situations, although it could be worse if starvation occurs, the resulting maximum throughput will be a lower bound. In other words, we can expect the actual throughput to be higher than ρ_{random} .

Maximum throughput based on the burst access bus usage

The analysis of the maximum throughput based on the burst access bus usage is somewhat simpler than that based on the random access bus. Burst access is used in cases where blocks of continuous memory locations are accessed together. The maximum throughput can be computed by estimating the number of data movements required. The following notations are used in deriving the formula.

β = bandwidth of burst access

- m_{up} = number of packet data movement in upward processing
 m_{down} = number of packet data movement in downward processing
 μ_{burst} = average memory cycle time for burst access
 w = bus width (in bits)
 x = percentage of traffic that is from the net (upward)
 ρ_{burst} = maximum processing throughput derived from burst access bus usage

$$\begin{aligned}
 \rho_{burst} &= x \frac{\beta}{m_{up}} + (1 - x) \frac{\beta}{m_{down}} \text{ MBytes/s} \\
 \text{where } \beta &= \frac{1}{8} \frac{w}{\mu_{burst}}
 \end{aligned} \tag{2.4}$$

The maximum throughput derived here gives an upper bound on the data output rate since we are ignoring any other usages that may be required. For instance, more than one access is needed to access a block of data which crosses the memory device boundary.

Estimation of the maximum throughput

The actual maximum throughput that can be obtained depends on the maximum throughput obtainable from both access modes and also the processing power available. The relationship between the maximum throughput and the processing power is simple because a certain amount of instructions are needed per byte of packet data processed. Therefore, we have:

$$\rho_{max} \leq \rho_{processor}$$

Since there is a complex relationship between the two access modes ⁴, we can only use some kind of heuristic to estimate a value for the throughput.

When $\rho_{random} \gg \rho_{burst}$, the maximum is approximately equal to ρ_{burst} . When $\rho_{burst} \gg \rho_{random}$, the maximum is approximately equal to ρ_{random} . When both values are

⁴when the burst access bus usage increases, we get a higher throughput and this implies more shared information access. The converse is true.

about the same, an upper bound of the maximum would be equal to the $\min[\rho_{burst}, \rho_{random}]$.

In other words, we have

$$\rho_{bus} \simeq \begin{cases} \rho_{burst} & \text{when } \rho_{random} \gg \rho_{burst} \\ \rho_{random} & \text{when } \rho_{burst} \gg \rho_{random} \\ \min[\rho_{burst}, \rho_{random}] & \text{when } \rho_{burst} \simeq \rho_{random} \end{cases}$$

Since $\rho_{processor}$ gives the maximum processing throughput that could be provided by the processors assuming that all other resources are available and ρ_{bus} gives the maximum processing throughput that could be supported by the buses assuming that all other resources are available, the maximum processing throughput can be estimated by

$$\rho_{max} \simeq \begin{cases} \rho_{processor} & \text{when } \rho_{processor} < \rho_{bus} \\ \rho_{bus} & \text{when } \rho_{processor} > \rho_{bus} \end{cases} \quad (2.5)$$

In other words, if the processors can process more data than the bus can support, the maximum will be limited by ρ_{bus} since there is no way for the system to keep the processors busy all the time. The converse is true because if we do not have enough processing power, the throughput will be limited by $\rho_{processor}$ as the bus will not be fully utilized.

2.5.5 Other parameters for measuring the performance

In addition to optimizing the maximum processing throughput, we would like to maintain a high resource utilization. The resource utilization is a function of many parameters. In estimating the resource utilization, we have to work out the details of the locking mechanism described in section 2.5.3. We also have to deal with the changes in the system load when packets are suspended and reactivated. We also need to consider the different complexities in transfer syntax conversions which vary with the data in the packets. Finally, the speed and number of processors in the system and their effects on resource contention have to be included as well. Since a large number of parameters have to be taken into account, the analysis becomes very complicated and hard to track. In our analysis, we have instead relied

on simulations to give us an estimate of the resource utilizations because it is easier to include the parameters mentioned above into the simulation program. We are mostly interested in the processor and bus utilizations which are defined below. Processor utilization is defined here as the ratio of the maximum throughput per processor in the multiprocessor system over the maximum throughput in a single processor system. This value gives us an estimate of the amount of processing power used in synchronization activities and lost in bus contentions in a multiprocessor system. Bus utilization is defined as the ratio of time the bus is used over a period of time. This value simply measures how much of the bus is being used and is related to the memory access cycle time. If memory is slow, each transaction will take a longer time and for the same number of accesses, the bus utilization will be higher. Other parameters like the number of packets processed or the average processing latency are also collected and reported.

2.5.6 Other major components

A protocol processing system consists of a number of other essential components in addition to the processors, memory devices and the interconnection structure. These components have to deal with the interfacing tasks between the host and the system and between the network and the system. They are also used to speed up certain critical operations like calculating the checksum and implementing timers.

Host interface

The host interface is responsible for communicating with the host machine. Since our protocol processing system is designed as a peripheral to the host computer, this interface must be capable of working with the I/O port of the host. One of its major functions is to ensure that data are buffered in both directions to minimize data loss. In our analysis, only a high level design is used and the actual design, which is machine specific, is not discussed.

Network interface

The network interface is used by the system to communicate with the network. It is responsible for taking care of the sending and receiving of data through the network. It could be combined with the network adapter to form a single unit. In that case, specialized circuitry would be needed based on the protocol used in the physical layer. For an ATM network, the interface would have to carry out the reassembly of cells and perform the checksum calculation. As in the case of the host interface, we only use a high level model for the network interface. Studies concerning the design of the network adapter can be found in [29] and [10].

Checksum unit

Checksum calculation is needed in the transport layer for reliable communication. Since checksum computation involves all the data in the packet, it is a very computation intensive operation. Using a specialized circuit for doing the calculation could improve the overall performance of the system [37] at a fairly low cost with a VLSI implementation. Processors in the system can then be freed to perform other tasks which are not as straight forward.

The checksum unit comprises of a specialized adder circuit that can compute the checksum when the data is sent through it in a stream using the burst access bus. The actual number of checksum units will vary according to the design. Since there is only one burst access bus, a single checksum unit may seem to be sufficient. However, if the result is returned through the random access bus, then the tradeoff of using multiple unit versus random access bus contention must be considered. In some of our designs, it is found that multiple units are more efficient than a single unit.

Timers

Many timers are used in protocol processing especially in the transport layer for ensuring a reliable transfer of data. The processing cost for implementing timers can be substantial [21]. Depending on the method of implementation, most of the processing can either be lumped

together in the phase for setting the timer or spread out in the phase for maintaining the timer. In the former case, an ordered list of the time remaining before expiry of the timers is maintained and processing is needed when inserting a new timer. In the latter case, no special processing is needed for setting the timer but processings are needed in counting down each of them. In our design, we have taken the former design and used a specialized processor to take care of all the timer processings. In our study, we are only interested in maximizing the processing throughput and error or flow controls are thus not considered. Therefore, in terms of simulating timers, we have restricted ourselves to look at the cost of setting them up but not in the detailed operations when a timer expires.

DMA device

To facilitate the movement of large blocks of data on the burst access bus, DMA devices are needed. Since all processors and the host and network interfaces are capable of initiating a block transfer for moving packets, each one of them is equipped with a DMA unit to reduce the need for using the random access bus and shared memory, which will normally be utilized if only one single DMA unit is installed in the system.

Bus Arbiter

Since the two shared buses are the major bottleneck of the system, we need an efficient scheme for allocating these resources. The First Come First Serve arbitration scheme is chosen to give the best possible performance [1]. Specialized hardware is needed to make sure that bus arbitration could be performed in parallel with bus transactions to maximize the use of the buses. Other simpler arbitration schemes are not used because of the importance of performance here.

2.6 The simple shared memory design

A simple shared memory (SSM) design based on the discussion in [37] is used to see if the simplifying assumptions made in the analysis are acceptable. The result is also used as a basis for comparison with other designs to be discussed later. The design of the system is shown in figure 2.9. It consists of all the major components described above and they are connected together using the two shared buses.

Figure 2.9: Block diagram of the SSM design

In this system, there is no local storage and both instruction and data are stored in the shared memory. Since everything is stored in the shared memory, the complete packet is moved only a few times during the processing. The number of movement is different when the packet is moved up and down the protocol stack as shown in figure 2.10. For upward processing, once the packet enters the system, it will be moved by the network interface

into the shared memory where most of the processing is performed. The packet is moved through the checksum unit during transport layer processing for error detection and is moved a third time to the host interface when all processing is done. For all other processing steps, the packet remains in the shared memory and is being accessed word by word through the random access bus. In total, three packet movements are required for upward processing.

Figure 2.10: Number of times a packet is moved during processing

For downward processing, the situation is similar except that the packet does not need to be moved through the checksum unit in a separate step. Instead, the checksum can be calculated and inserted into the packet when the packet is being moved from the shared memory to the network interface before it is being sent out into the network. The number of data movements is thus reduced to two. This modification reduces the number of data movements but may be regarded as a violation of the protocol stack model [40].

The expected throughput is determined by making the following assumptions. Hit

ratios as shown in table 2.3 are used. Here we are assuming that buses are 32 bits wide. For

access category	Transfer Syntax conversions	Header processings	Locks access	Header data access	Packet data access
hit ratio	0.975	0.975	0	0.75	0.75

Table 2.3: Hit ratios for simulating the SSM design

instruction accesses, we have assumed that about 75% are register only instructions⁵ and that the hit ratio of these is 1. For the remaining 25% of instruction accesses, we have assumed a hit ratio of 0.9. Thus on average, the hit rate is 0.975 for all instruction accesses. To simplify the analysis of synchronization operations, we have assumed that the hit ratio for state and queue access to be 0. If we have assumed a non zero hit rate, we would need to consider data cache and cache coherence schemes which will complicate the analysis because of the extra coherence traffic on the shared buses. For data access, we have assumed that all the data fetched will be used and since each time 4 bytes are fetched with a 32 bit bus, a data hit rate of 0.75 is obtained. This value is a little optimistic but since transfer syntax conversion may touch a large portion of the packet data, the error introduced is small.

The lock retry interval α is assumed to be a fixed value of 100ns. From the simulations, it was found that the lock holding time $y \simeq 10\alpha$ for most situations. Therefore, from equation 2.1 the number of retries can be written as $r \simeq \frac{(n-1)11}{2}$.

Other parameters used are shown in table 2.4:

parameter	μ_m	μ_e	l	p_n	γ	x	μ_{burst}	m_{up}	m_{down}
values	200ns	100ns	10000	4096	17	0.5	50ns	3	2

Table 2.4: Parameters used for simulating the SSM design

Substituting these values into equation 2.3

$$\rho_{random} \simeq \frac{10000}{300 * 10^{-9}(6028 + 24.42n)} * 8 \text{ MBits/s}$$

Thus for $n = 1$, $\rho_{random} = 44.08 \text{ MBits/s}$ and for $n = 20$, $\rho_{random} = 40.80 \text{ MBits/s}$. If there is more than one connection, the amount of lock contentions would be lowered and thus there

⁵Estimation of the ratio of register only instructions is discussed in appendix A

will be less retries and a higher throughput is possible. From equation 2.4, the maximum throughput that could be supported by the burst access bus is:

$$\rho_{burst} = 264\text{MBits/s}$$

Since $\rho_{burst} \gg \rho_{random}$, the bottleneck is at the random access bus and the maximum throughput would be closer to 40.80MBits/s than to 264MBits/s. Since $\gamma = 17$, the processing power required is around 85mips which can be obtained with eight to nine 10mips processors.

Figure 2.11: Analytical and Simulation results of the throughput for the simple shared memory design

The analytical and simulation results of the maximum throughput are shown in figure 2.11. The simulation result follows $\rho_{processor}$ up to about 8 processors. Thus linear

increase in throughput is only achieved with 8 processors. When more processors are added, the throughput saturates at about 44MBits/s as predicted by ρ_{random} .

From the graph of bus utilization in figure 2.12, we can observe that the random access bus is a major point of contention. Since all instructions and data accesses have to go through this bus, it saturates quickly with only about twelve 10mips processors. On the other hand, the burst access bus is hardly used. This also explains why the ρ_{burst} , which is based on the burst access bus usage, is way off in estimating the maximum throughput. The simulation result shows that our analysis is quite accurate in describing the trend and in estimating the throughput to within 10%. We also did not observe any deadlock occurring with the simple locking mechanism.

The simulation is repeated using 20 and 40mips processors. The results are shown in figure 2.13. With higher speed processors, we could provide the same processing power with a fewer number of processors than by using 10mips processors. This would then reduce the amount of bus contentions due to lock contentions and allow us to increase the maximum throughput. However, the simulation result indicates that high speed processors do not help very much in improving the maximum throughput. The maximum throughput as obtained by using 40mips processors is only about 32% more than that of the 10mips processors. Thus the return for using faster processors is very low. The main reason for this result is that the bottleneck here is not caused by the lock accesses which would increase with the number of processors. Instead, the bottleneck is caused by the high amount of transfer syntax operations that are carried out using the random access bus. Therefore, the effects of having to busy wait for a lock, which increases with the number of processors, are not observed because the processors spend most their time waiting for the bus doing transfer syntax conversions rather than accessing the shared states.

From our results, we can conclude that using the SSM design to implement the single active queue processing model is inefficient and more sophisticated designs are needed.

Figure 2.12: Bus utilizations of the random access and burst access bus for the SSM design

Figure 2.13: Throughput and bus utilization of the SSM design with 20mips and 40mips processors

Chapter 3

The local instruction memory design

In this chapter, we are going to look at some design features for improving the system performance. From the result of the SSM design, we can observe that even though the instruction hit ratio is high (at 0.975), instruction accesses still create a lot of shared memory references because of the high proportion of transfer syntax operations which is in the order of twenty or more instructions per byte of packet data. One effective means of improving the system performance is to make use of locality to reduce the shared memory access. If the information needed is easily accessible without accessing the shared resources, the speed of the operation can be increased. In the following discussions, we are going to see how we could apply the effects of locality to the design of protocol processing systems.

3.1 Background on locality

There are two aspects to the property of locality, namely the spatial and temporal aspects [11]. Temporal locality refers to the property that information which will be used in the near future is likely to be in use already. This type of behavior is observed in program loops where instructions and variables are reused. Spatial locality is the property that the location of the information to be accessed will be near to the ones that are being currently used. This is seen mostly in program codes and large data structures where data are accessed sequentially.

In an application, we could divide the memory references into instruction access and data access. Instruction access has a higher degree of locality when compared to data access in general because of the sequential execution of program codes and the various loop constructs. For data access, the number of times that a piece of data is used and the location of the data

used depend a lot on the application. In general, the degree of locality is lower than that of instruction access. In case of protocol processing, the main data structure is the packet data. If we consider only a single packet, temporal and spatial localities can be found since both the presentation and transport layer would potentially touch all the bytes in the packet during processing. However, if we consider the process in general, then little spatial and temporal locality can be obtained as packets keep coming in and moving out of the system at a high rate.

A lot of work have been done in improving cache system performance by prefetching when the inherent locality is not enough. Prefetching techniques range from simple ones which only prefetch the next memory location to ones which are controlled by program code. In our research, we are going to apply the prefetching technique to reduce the shared memory access by moving the program code and packet data closer to the processor into local storages before they are needed. This could be done because the packet is used heavily only in the presentation and transport layer and we know when these are performed. We do not simply apply prefetching to our cache because doing that alone is not enough in removing shared memory access which is a main problem in our design. In this chapter we will study the effect of keeping instructions locally and we will look at the effects for packet data in the following chapter.

3.2 Effects of locality

For the shared memory design discussed in the last chapter, both instructions and data are stored in the shared memory. In the transfer syntax conversion phase, even for a simple data structure like the personnel record used in the ASN.1 specification, 20 to 40 instructions per byte of packet data may be required for doing transfer syntax conversion using the basic encoding rules [37]. Fetching these instructions from the shared memory creates a lot of traffic on the shared bus and limits the total system throughput because of bus contention. Using the shared memory for temporary variables further adds to the problem as they create more

shared bus traffic.

The result of our simulations for the SSM design shows that the random access bus usage rises quickly with the increase in the number of processors while the burst access bus usage is low. Our first step in studying the effects of locality is to see the change in performance when instructions and temporary variables are stored locally instead of in the shared memory so as to reduce the amount of random access bus traffic. This change is implemented by adding local memory to each of the processors. The local storage is designed to hold all codes and temporary variables required for processing. With this design, the shared memory is only used for storing connection state, system state and packet data. If we use the analogy that our shared memory is like the main memory in a hierarchical memory system and that the local memory is the cache, we would have a perfect hit ratio here in terms of instruction access. We call this architecture the local instruction memory (LIM) design.

The architecture of the LIM design is essentially the same as that of the SSM design. The only difference is the addition of local memory to each of the processors. The local memory is connected to its processor through the random access bus of the processor and is only accessible by that processor as shown in figure 3.1. Since the local memory is only available to one processor, no special arbitration hardware is needed. The design could be implemented by using the supervisor and user address spaces found in most common general purpose microprocessor or by using a memory mapped address space. The major cost of the design comes from the memory chips.

3.3 Performance analysis and Simulations

The way in which packets are processed is the same as the SSM design. Since we are only changing the location of the code and temporary variables and this is all done in hardware, the software would be essentially the same as the SSM design. In the analysis for the SSM design, the hit ratios refer to memory accesses to the shared memory. With local memory installed in the system, some of the hit ratios have to be modified. The numbers are changed

Figure 3.1: The Local Instruction Memory design

as shown in table 3.1. They are only used in deriving ρ_{random} . The same hit ratio as in the

access category	Transfer Syntax conversions	Header processings	Locks access	Header data access	Packet data access
hit ratio for LIM	1.0	1.0	0	0.75	0.75
hit ratio for SSM	0.975	0.975	0	0.75	0.75

Table 3.1: Hit ratios for simulating the LIM design

SSM design is used for the local memory in the simulation so that instructions stored in local memory would still have a hit rate of 0.975. However, fetching these instructions from the local memory would not add any contention to the shared bus. We can obtain a good estimate of the system behavior without being too optimistic by using these hit ratios although a hit rate of closed to 1 is achievable if we have a perfect instruction pipeline implemented with cache prefetch mechanisms like branch prediction [31], stream buffers, victim buffers [24] and software prefetch schemes [6] [26].

Using the same assumptions and parameters used for the simple shared memory design, we have:

$$\rho_{random} \simeq \frac{10000}{300 * 10^{-9}(1724 + 24.42n)} * 8 \text{ MBits/s}$$

which is a function of the number of processors

As n increases, there will be more contentions for the bus and this will result in a lower throughput. Since packets are processed in the same way as before, the number of packet movements is the same and so ρ_{burst} remains unchanged or

$$\rho_{burst} = 264 \text{ MBits/s}$$

Even for $n=1$ where there is no contention, $\rho_{random} = 152.48 \text{ MBits/s} < \rho_{burst}$. From section 2.5.4, we estimate that the maximum throughput is limited by ρ_{random} .

The simulated throughput is plotted against the number of processors as depicted in figure 3.2. The analytic results derived for ρ_{random} , ρ_{burst} and $\rho_{processor}$ are also plotted on the graph. The simulated throughput follows $\rho_{processor}$ up to about 30 processors and starts to saturate to about 112 MBits/s which is just a little larger than the ρ_{random} . Thus we could obtain a 3 times increase in maximum processing throughput as compared with the SSM. A linear increase in throughput with the number of processors is maintained until saturation occurs and the bus design is found to be capable of supporting 3 times more processors in this configuration than the SSM design.

The utilization of the burst access bus is also improved as shown in figure 3.3. By comparing with the bus utilization of the SSM design, the difference in the utilization of the two buses in the SSM and LIM designs has been reduced from approximately 0.8 to 0.45. The increase in burst access bus usage is directly related to the increase in throughput. From equation 2.4, we can see that burst access bus usage increases linearly with the throughput. By storing instructions and temporary variables in the local memory, we are able to reduce the traffic on the random access bus and allow the system to utilize more processors and thus handle more packets.

Figure 3.2: Analytical and Simulation results of the throughput for the LIM design

Figure 3.3: Bus utilizations of the random access and burst access bus for the LIM design

The processor utilization is found to be better than 0.9 for most of the time before saturation of the bus occurs. Together with the linear increase in throughput, we can see that the overhead introduced by parallel processing is small as most of the processing power goes toward producing a higher output as desired.

The simulations are repeated with 20mips and 40mips processors with all other parameters remaining the same. The results are shown in figure 3.4. By using faster processors,

Figure 3.4: Throughput and bus utilization of the LIM design with 20mips and 40mips processors

we can obtain the same processing power with a fewer number of processors. When the number of processors is low, the amount of bus contention will also be low and therefore more

processors can be added. This would imply that a higher throughput can be achieved. The simulation results supported this argument. For the same throughput, say at 100MBits/s, the random access bus usage of the 20mips system is higher than that of the 40mips system as shown in figure 3.4. As a result of the higher bus utilization, a higher bus contention will be seen in the 20mips system and the maximum throughput achievable on the 20mips system will be lower than that of the 40mips system.

Table 3.2 shows the maximum throughputs obtained from the SSM and LIM design with different processor speeds. The results indicate that with the LIM design, we can increase

processor speed (mips)	SSM design		LIM design	
	max throughput MBits/s	% increase with 40mips cpu	max throughput MBits/s	% increase with 40mips cpu
10	43.52	32	109.60	50
20	52.48	10	143.12	15
40	57.60	0	164.72	0

Table 3.2: Variation of maximum throughput with processor speed

the maximum throughput more effectively by using higher speed processors. As discussed in section 2.4, using high speed processor without high speed memory would not be efficient and the results here illustrate that point. When the processor speed is increased by 4 times, the maximum throughput is only improved by 50% in the best case when the memory speed is kept the same.

3.3.1 Merits of the LIM design

The LIM architecture is designed to tackle some of the problems in the SSM design. There is only a minor modification in the hardware with the inclusion of local memory for each processor with minimal software changes. The use of local memory for storing program codes and temporary variables removes a big portion of memory accesses to the shared memory. Although there is not a lot of changes in the design as compared to the original, the maximum processing throughput is almost tripled for the different processor speeds simulated. The

number of processors that can be accommodated and still maintaining a linear speedup is also increased by more than 3 times. The utilization of the burst access bus has been increased while that of the random access bus has been reduced to obtain a more balanced use of the shared resources.

3.3.2 Weakness of the LIM design

A major disadvantage of this design is that instructions have to be duplicated in the local memory of each processor. Thus in terms of instruction memory usage, the efficiency is proportional to $\frac{1}{\text{number of processors}}$. This is not a major problem because the cost of memory is expected to go down with the advent of technology. Another problem with this design is that false sharing of packet data is not addressed. Since there is only one access path to the packet data in the shared memory, whenever a processor is accessing a packet, no other processor can use the shared memory even if it needs to access a different packet. Assuming that only 1 out of 4 bytes of the packet needs to be accessed during transfer syntax conversion, there is still a lot of memory traffic. This problem seriously affects the system throughput and is dealt with in the next chapter.

Another undesirable characteristic of the design is that there is a low utilization of the faster burst access bus and a high utilization of the slower random access bus. The random access bus is thus the bottleneck again as in the SSM design. When the random access bus saturates, there is still a lot of bandwidth remaining on the burst access bus. This reveals that some improvement is possible without major modification to the hardware if we can change the usage of the buses.

Chapter 4

The packet relocation design

In the discussion of the LIM design, we have noticed that a fair amount of random access traffic is still required because packet data are stored in the shared memory. The resulting utilization of the slower random access bus is higher than that of the burst access bus and the processing throughput is limited to a lower value than allowed by the faster bus. In this chapter, we would look at an improvement that attempts to reduce the shared memory access by moving packets to the local memory. We could also balance the load on the two buses by using the burst access bus for the relocation of packets.

4.1 Usefulness of data cache

Since we know when the packet data will be used during protocol processing, we could prefetch the packet data into a local cache of the processor using a cache prefetch technique. However, we still have to use the shared random access bus to perform this prefetch, which is carried out like a cache miss operation, and therefore the overall bus contention problem is not improved. Since a cache prefetch instruction can only bring in a certain number of bytes determined by the bus width and the packet size varies a lot, extra calculations would be needed to determine how many prefetches are required and where and when to issue them. This could complicate the processing and affect the processing latency. Moreover, packet data are not reused extensively and thus leaving them in the cache would not be useful.

Instead of using a cache prefetch mechanism, we choose to simply move the data from the shared to the local memory just before the access begins. The advantage is that local memory could be made to support burst transfer and data can reside in them for a longer

time without the possibility of becoming invalid like in the cache. By using the burst access mode to move the data, only one single transfer is needed to relocate the data and the random bus is not used.

4.2 Spatial locality achieved through copying

To see why moving packets to the local memory will improve the performance, we have to look at how packet data is used during protocol processing. In the transfer syntax conversion phase, representation of the packet data has to be changed from one form to the other. For instance, all of the bytes have to be read and rewritten for the integer data type. If the packet is stored in the shared memory, the processor has to get hold of the bus, access a few bytes, release the bus and repeat the cycle again until the whole packet is processed. If there is more than one processor in the system, bus contention would occur and slow down the processing.

Since the transfer syntax conversion of a packet is considered as a single step and is carried out by one processor, the packet needs not be shared. Therefore, if the packet is moved to the local memory before the transfer syntax conversion phase begins, the false sharing of packets will be reduced and the shared random access bus traffic will also be lowered. The key idea here is to use the burst access bus instead of the random access bus to do the transfer. By doing so, we can also effectively balance the usage of both buses. We call this architecture the simple packet relocation (SPR) design.

The system architecture of the SPR design is similar to the LIM design. However, in this design, the local memory is dual ported, one for random and one for burst access as in the global shared memory. The random access port is only connected to the local processor through the local random access bus while the burst access port is connected to the shared burst access bus as shown in figure 4.1. Since all the local memories are connected to the shared burst access bus, the memory system could be viewed as a distributed shared memory system. Dual port memory devices are used to facilitate the movement of data from shared to local memory and the access to code and data by the local processor.

Figure 4.1: The Simple Packet Relocation design

The processing steps are essentially the same as before with the addition of some extra data movements as shown in figure 4.2. For upward processing, global shared memory

Figure 4.2: Number of packet relocations for the SPR design

is used for storing the packets up to and including the session layer. In the presentation layer where there is potentially a high number of packet data accesses, the packet is copied from the shared memory to the local memory using block transfer on the burst access bus. By prefetching the data to the local memory, we have made a tradeoff between the decrease in random access bus usage with the increase in burst access bus usage. Packets are left in the shared memory for the lower layers because of the need to perform reassembly operations which can be carried out more efficiently when all packets are in the same address space so that no copying is required.

For downward processing, packets are copied from the shared to the local memory of the processor when processing begins. This packet can stay in the local memory of the

processor which would carry out all the processings if possible. In an ideal situation, when the sequence number is correct for all connection oriented layers, the packet would be kept in the local memory of the processor until the packet is sent out. In a less than ideal case, when the sequence number is incorrect, the packet will be copied back to the shared memory. This is done so that the current processor is freed to work on something else and that keeping the packet in the global shared memory allows any processors to work on it at a later time.

4.3 Performance analysis and Simulation

Since packets have to be moved to the local memory, the software has to include proper DMA instructions. Each processor is equipped with a DMA device so that each processor can initiate a packet transfer. Other than that, the software and hardware would still be very similar to the LIM design. Since the packet is now in the local memory for the transfer syntax conversion, the hit ratio for accessing the packet data has to be changed. The new hit ratios are shown in table 4.1. As in the case for the LIM design, these hit ratios refer to the shared

access category	Transfer Syntax conversions	Header processings	Locks access	Header data access	Packet data access
hit ratio for SPR	1.0	1.0	0	0.75	1.0
hit ratio for LIM	1.0	1.0	0	0.75	0.75
hit ratio for SSM	0.975	0.975	0	0.75	0.75

Table 4.1: Hit ratios for simulating the SPR design

memory. For items which are now stored in the local memory, the hit ratio is always 1 as far as the shared memory is concerned as the shared memory would not be accessed. For the local memory, the original hit ratios used for the simple shared memory design will be used.

Using the same assumptions as before, we can derive the maximum throughput using equation 2.3:

$$\rho_{random} \simeq \frac{10000}{300 * 10^{-9}(474 + 24.42n)} * 8\text{Mbits/s}$$

For $n = 1$, $\rho_{random} = 534.84\text{Mbits/s}$ and for $n = 40$, $\rho_{random} = 183.84\text{Mbits/s}$. Depending on

whether we need to copy the packet back to the shared memory during downward processing, we would have a best and a worst case for ρ_{burst} . For the best case, only 3 data movements are needed for downward processing ($m_{down} = 3$). For the worst case, 4 movements ($m_{down} = 4$) are required.

$$\begin{aligned}
 \text{Best case: } \rho_{burst} &= \left(\frac{0.5 * 80 * 10^6}{4} + \frac{0.5 * 80 * 10^6}{3} \right) * 8 \\
 &= 186.64 \text{ Mbits/s} \\
 \text{Worst case: } \rho_{burst} &= \left(\frac{0.5 * 80 * 10^6}{4} + \frac{0.5 * 80 * 10^6}{4} \right) * 8 \\
 &= 160 \text{ Mbits/s}
 \end{aligned}$$

In this design, ρ_{burst} is smaller than ρ_{random} in both the best and worst case for a large range of values for the number of processors. Therefore, in this design, the burst access bus is the bottleneck of the system. The maximum processing throughput would be around 160 Mbits/s and the number of 10mips processors required is around 42.

Both analytical and simulation results are plotted as depicted in figure 4.3. The simulated throughput follows the analytical value closely and saturates at around 144 Mbits/s. The number of processors that can be accommodated is almost five times higher than that of the SSM design. The utilization of the burst access bus is higher than that of the random access bus as predicted by the analysis. From the plot in figure 4.4, we can observe that the utilization of the two buses have been interchanged as compared to the previous designs. A strange behavior is observed when the number of processors rises to about 45 when the random access bus utilization increases sharply and the burst access bus utilization drops. As the burst access bus usage decreases, the throughput also drops. This observation could be explained in terms of the contention in the random access bus. When there is a high number of processors in the system, the system may become unstable if all the processors try to compete for some shared resources. A good guideline would then be to keep the number of processors to around 35 to 40 under most situations.

The erratic behavior is not observed when the processor speed is increased to 20mips

Figure 4.3: Analytical and Simulation results of the throughput for the SPR design

Figure 4.4: Bus utilizations of the random access and burst access bus for the SPR design

and 40mips as shown in figure 4.5. With faster processors, a fewer number is needed to

Figure 4.5: Throughput and bus utilization of the SPR design with 20mips and 40mips processors

provide a certain processing power and this can reduce the probability of bus congestion. The maximum throughput obtained with different processor speeds are very similar as observed from the simulation result. In other words, the maximum throughput is not very sensitive to the speed of the processors used. This behavior is similar to that of the simple shared memory design but the reasoning is different. In the former case, there is so much random access bus traffic due to transfer syntax conversion that the effects of increasing the processor speed are not significant. In the SPR design, the bottleneck is in the burst access bus and

the traffic on which does not depend on the number of processors nor the processor speed. A better processor utilization trend is thus possible with the SPR design. Even when the burst access bus saturates, the processor utilization is still better than 0.85. Comparing with the LIM design in figure 3.4, the processor utilization near saturation drops much faster in the LIM design than here.

4.3.1 Merits of the SPR design

The SPR design reduces the false sharing problem and shifts the bottleneck to the faster bus by adding dual ported VRAMS as local memory and some minor changes to the software. The number of processors that can be supported in this design is almost 5 times that of the SSM design with the maximum throughput being around 4 times higher. The burst access bus utilization is improved significantly and becomes the throughput controlling factor of the system. In practice, burst access is more than 4 times faster than random access in dual ported VRAMS [17] as those used in our simulations. With a burst access time of 30ns, ρ_{burst} would be increased to 267Mbits/s. The advanced design in cache DRAM [4][5] suggests that we have the technology to construct VRAMS with even faster burst and random access which will further improve the performance of this design. Another improvement that we have achieved is that the maximum throughput of the system is not dependent on the speed of the processors. Therefore, the design allows us to use low speed processors, which could reduce the cost of the system, without lowering the processing throughput.

4.3.2 Weakness of the SPR design

When the number of processor increases, the system can become unstable when the random access bus usage suddenly rises. As more and more packets are processed per second, the demand for the state information and other synchronization data also increases and would cause the random access bus to become the bottleneck again. Thus a further reduction in random access bus usage is needed to ensure that more packets can be processed when faster

memory devices become available especially for a faster burst access cycle time.

The success of this design depends on a fast burst access mode. If burst access is slow because of slow memory or frequent crossing of memory device boundaries, the improvement would decrease. We could compare the SPR design which is limited by the burst access speed with the LIM design which is limited by the random access speed. With a 32 bit bus, equal network and host traffic and 4 copies for both upward and downward processing, ρ_{burst} for the SPR design reduces to $\frac{1}{\mu_{burst}}$. Table 4.2 shows the change in ρ_{burst} for this design when the speed of burst access is varied.

μ_{burst} ns	25	50	100	150	200
ρ_{burst} MBits/s	320	160	80	53.33	40

Table 4.2: Variations of maximum throughput when the burst access speed is changed

Plotting the data in table 4.2 with ρ_{random} for the LIM design based on a 200ns random access cycle time , we can see from figure 4.6 that when the burst access cycle time μ_{burst} is more than $\frac{1}{4}$ that of the random access cycle time, no benefit can be derived from this SPR design. Therefore, when implementing the system, the memory management component must be designed carefully to reduce the possibility of storing a packet over a few memory devices so as to make sure that burst access is fast enough.

The extra data movements involved in this design increase the latency time especially for the situations in which very few data movements are needed for transfer syntax conversion as in character strings. To see if this is a real concern, data have been collected to estimate the latency time. Tables 4.3, 4.4 and 4.5 show the average processing latency time as obtained from the simulations.

The average bus request delay is also shown. Assuming that the average length of a packet from the host is 10000 bytes, the average copying time with a 50ns burst access cycle and a 32 bit bus is about 125 μ s. From the simulation results, we can see that the copying time and the bus delay is about 2 orders of magnitude smaller than the processing time. Thus, the increase in latency due to the extra data relocation would not be significant.

Figure 4.6: Comparison of the LIM design and the SPR design with different burst access speeds

number of processors	processing time per packet (ms)	# of packets processed	# of packets copied back	% of packets copied back	bus request time(ns)
1	20.3	23	0	0	9689
2	23.6	39	3	8	11803
4	21.9	85	18	21	13718
8	22.6	170	54	32	21416
16	23.2	327	130	40	35388
20	22.4	415	181	44	53157
32	22.0	654	299	46	96507
40	22.4	783	386	49	131078
48	27.1	626	343	55	127694

Table 4.3: Average processing time using 10mips processors in a 0.5s interval

number of processors	processing time per packet (ms)	# of packets processed	# of packets copied back	% of packets copied back	bus request time(ns)
1	12.5	38	0	0	11597
2	11.9	81	3	4	16453
4	12.6	151	25	17	19961
8	12.7	301	86	29	26712
16	12.3	604	239	40	69417
20	12.6	728	306	42	104359
26	12.9	890	423	48	158769
27	12.9	921	451	49	160492
28	13.2	942	498	53	176760

Table 4.4: Average processing time using 20mips processors in a 0.5s interval

number of processors	processing time per packet (ms)	# of packets processed	# of packets copied back	% of packets copied back	bus request time(ns)
1	6.9	70	0	0	12784
2	6.9	138	4	3	17498
4	7.2	264	35	13	25651
8	7.2	520	141	27	49501
12	7.2	758	263	35	80958
14	7.3	856	299	35	101947
15	7.5	890	342	38	124350
16	7.6	932	362	39	143348
17	7.4	1002	576	57	192487
18	7.2	1088	797	73	264338

Table 4.5: Average processing time using 40mips processors in a 0.5s interval

The need to move the packet from local to shared memory when they are out of sequence is also a concern. Table 4.3, 4.4 and 4.5 also give the percentage of packets that have to be moved because of the serialization requirement. When the number of processor increases, the need to copy packets back to shared memory also increases. Over 50% of the packets have to be moved when the system is closed to saturation in all cases simulated. These extra copies result in a severe overhead which cannot be neglected.

4.4 Overcoming the weakness

Architectural changes are made to deal with the problems described in the last section and to further improve the system performance. We would like to reduce the bus utilization, to extend the maximum processing throughput to a gigabit rate and to maintain a graceful degradation of performance where there is a sudden increase of workload owing to bursty traffic or failure of some of the processors. The new architecture is called the improved packet relocation (IPR) design.

4.4.1 Reduce the number of packet relocation before processing starts

For the SPR design, packets buffered in the host interface for downward processing are copied into the shared memory when the packets arrive. The packet is copied again into the local memory of some processor when the presentation layer processing starts¹. The reason for doing that is to give every processor equal access to the data so that when a processor is ready, it can process any packet in the shared memory. This arrangement results in an indirect copying step which uses up extra bandwidth on the burst access bus and the shared memory. We can eliminate the copying from the interface to the shared memory by moving the data directly from the interface to the local memory of the processor concerned. The detailed operation is discussed below.

When a packet enters the system from the host, the active queue is updated as before.

¹refer to discussions in section 4.2

However, the packet remains in the host interface buffer to minimize the use of the shared buses. When a processor becomes free, it gets the first job from the active queue and if that is a new job from the host, the processor will initiate a transfer of the packet from the interface buffer directly into its local memory.

This configuration will also give the system more tolerancy to bursty traffic or failure of one or more processors. Since all packets are buffered in the interface, the shared resource usage will be proportional to the actual number of packets being processed and thus the system performance would not be affected by the number of jobs waiting in the active job queue. In other words, graceful degradation of performance is achieved.

For upward processing, the number of data movements cannot be reduced easily. The requirement for reassembling packets makes it efficient to leave packets in the shared memory. However, storing the packets in the interface buffer still helps in making the system more fault tolerant as in downward processing.

4.4.2 Reduce the number of packet relocation during processing

For downward processing, the data portion of the packet is not accessed after the presentation layer processing where syntax conversion is performed. Headers are simply added to the original PDU in the lower layers. Therefore, it is possible to avoid the need of copying packets back into the shared memory when they are out of sequence as done in the previous design. The saving can be substantial. If packets are long, copying them takes a lot of valuable resources. Even if they are short, it would still be a waste of resources as the header, which is the part really needed for lower layer processing, is usually much shorter than the data portion of the PDU. For instance, in an ATM cell where there is only 53 bytes, the data portion still consists of over 90% of the cell. The processor performing the processing could allocate the header in its local memory and update the state table with this information. When the packet is going to be sent out, instead of having 1 single transfer to move the data from the memory to the network interface buffer, multiple transfers are used to get the packet

and all the headers which may reside in local memories of different processors. In the worst case, each layer will be processed by a different processor and a total of five transfers would be needed. Since the length of the packet is not changed, usage of the bus would not be affected. However, there would be more overhead in terms of getting the bus and setting up the DMA transfers and the overhead introduced would worsen the processing latency time. On the other hand, the packet is moved only once in the IPR design rather than a couple times in the SPR design and therefore the processing time will be shortened because of the reduced number of data relocations. The actual effect on the processing time would have to be observed from the simulations.

Results of the IPR design

Figure 4.7 shows the number of data movement after making the above improvements. For downward processing, only 2 relocations are needed while for upward processing, 4 copies are still required. We can estimate the maximum throughput of this system using the analysis discussed in chapter 2. The hit ratios are the same and by using 40mips processors,

$$\rho_{random} \simeq \frac{10000}{225 * 10^{-9}(474 + 24.42n)} * 8 \text{ MBits/s} \quad (4.6)$$

When $n=1$, $\rho_{random} = 713.37 \text{ MBits/s}$ and when $n = 20$, $\rho_{random} = 369.44 \text{ MBits/s}$. The new value for ρ_{burst} is:

$$\begin{aligned} \rho_{burst} &= \left(\frac{0.5 * 80 * 10^6}{4} + \frac{0.5 * 80 * 10^6}{2} \right) * 8 \text{ MBits/s} \\ &= 240 \text{ MBits/s} \end{aligned} \quad (4.7)$$

Only 40mips processors are used for this design because the increased maximum throughput would require too many 10mips or 20mips processors which would exceed the physical limit that can be driven by the bus.

By reducing the number of data movements in downward processing from the average value of 3.5 to 2, we have increased the maximum throughput by over 25% from 176.96MBits/s

Figure 4.7: Number of packet relocations for the IPR design

to 227.20MBits/s as shown in figure 4.8. Therefore, the increase in DMA overhead for gathering the headers is really smaller than the decrease in data copying time. Figures 4.8 and 4.9 show the simulation and analytical results. The characteristics of the IPR design is very similar to that of the SPR design. High processor utilization is maintained until saturation of the burst access bus occurs. The introduction of the changes do not seem to result in any undesirable effects.

Figure 4.8: Throughput of the IPR design using 40mips processors

Table 4.6 gives the average processing time per packet. We can see that there is an improvement in latency time when compared to the SPR design as depicted in table 4.5. In other words, the extra overhead in setting up the transfer gives a smaller increase in time

Figure 4.9: Resource utilization of the IPR design using 40mips processors

number of processors	avg processing time(ms)/packet	# of packets processed
1	6.19	77
2	6.35	148
4	6.79	275
8	6.90	537
16	7.03	1016
20	7.35	1169

Table 4.6: Average processing time for the IPR design in a 0.5s interval

than the reduction in time resulting from the lower number of data movements. In fact, the amount of reduction is not important as long as there is no significant increase in the processing time as we are mostly interested in reducing the usage of the shared resources without increasing the latency time.

Since we are interested in a gigabit network rate, we would like to see what the system throughput would be with the fastest memory device and widest bus that is practical. The new system parameters are shown in table 4.7 and hit ratios in table 4.8.

parameter	μ_m	μ_e	l	p_n	γ	x	μ_{burst}	m_{up}	m_{down}	w
new values	190ns	25ns	10000	4096	17	0.5	30ns	3	2	64
old values	200ns	25ns	10000	4096	17	0.5	50ns	4	2	32

Table 4.7: Parameters used for finding the max throughput

access category	Transfer Syntax conversions	Header processings	Locks access	Header data access	Packet data access
hit ratio for IPR (wide bus)	1.0	1.0	0	0.875	1.0
hit ratio for SPR	1.0	1.0	0	0.75	1.0
hit ratio for LIM	1.0	1.0	0	0.75	0.75
hit ratio for SSM	0.975	0.975	0	0.75	0.75

Table 4.8: Hit ratios for finding the max throughput

With a 64bit bus, the hit ratio for state information is increased to 0.875 assuming that consecutive bytes fetched will all be consumed before they are removed from the cache. Therefore, we have

$$\rho_{random} = \frac{10000}{215 * 10^{-9}(437 + 24.42n)} * 8 \text{ MBits/s} \quad (4.8)$$

$$\begin{aligned} \text{and } \rho_{burst} &= \left(\frac{0.5 * 266 * 10^6}{4} + \frac{0.5 * 266 * 10^6}{2} \right) * 8 \text{ MBits/s} \\ &= 800 \text{ MBits/s} \end{aligned} \quad (4.9)$$

For $n=1$, $\rho_{random} = 806.41 \text{ MBits/s}$ and for $n=40$, $\rho_{random} = 263.18 \text{ MBits/s}$. The analytical results indicate that if the burst access speed is increased without a corresponding

increase in random access speed, the bottleneck could shift back to the random access bus. Since the calculation of ρ_{random} incurs a certain amount of uncertainty in estimating the number of attempts for obtaining a lock, simulation results are needed to see if the bottleneck will really shift back.

Maximum processing throughputs from analytical results and simulations are shown in figure 4.10. The maximum throughput is obtained with about 40 processors at 560Mbits/s.

Figure 4.10: Throughput of the IPR design using 40mips processors and fast memory

Linear increase in throughput is obtained for up to about 30 processors and processor utilization only drops to 85% at maximum throughput. The simulation results show that the maximum processing throughput lies somewhere between ρ_{burst} and ρ_{random} but not equal

to ρ_{random} as expected. Therefore, there is definitely some inaccuracy in the estimation of the number of lock attempts. The results also reveal that by speeding up the burst access cycle, a significant increase in throughput, which is not achievable with the SSM design, is obtained. The graph of resource utilization in figure 4.11 shows that the difference in the

Bus and processor utilization of the IPR design using high speed memory with 40mips processors

Figure 4.11: Resource utilization of the IPR design using 40mips processors and fast memory usage between the shared buses has reduced significantly. The burst access bus usage is only slightly higher than that of the random access bus until the system starts to saturate. This is a desirable characteristic because both buses are fully utilized and this is a major improvement as compared to previous designs.

To see if the system is fault tolerant as predicted, another test is run to observe the

effect of overloading the system. Packets are sent into a system with 4 processors at rates of 2 and 4 times that of the maximum throughput rate as determined earlier. A poisson distribution and exponential service time is used. The results, as shown in table 4.9, indicate that the maximum throughput remains relatively unchanged. Packets are stuck in the interface

system configuration	Throughput (Mbits/s)	Random access bus usage	Burst access bus usage	Avg Latency (ms)
peak value	64.41	0.096	0.096	5.88
input rate doubled	64.64	0.103	0.099	5.81
input rate quadruple	63.84	0.110	0.096	5.87

Table 4.9: System performance when the input rate is higher than the processing throughput

buffers and the buffer size required to store all the packets for 0.5s is between 8-24MBytes. The actual buffer size required would be dependent on the protocol's ability to regulate the traffic in reaction to a reduction in processing power. Work on congestion control and avoidance are discussed in [19][20]. Our simulation results indicate that there is a slight increase in random access bus usage. This is due to the need of accessing shared states and active queues when a new packet arrives.

4.4.3 Discussions

The IPR design provides a processing throughput of more than 500Mbit/s or half of that of a gigabit network. A balanced bus utilization is also obtained. However, the results reveal a number of weakness in the design and the analytical model.

By increasing the burst access speed alone, the bottleneck will go back to the random access bus. Thus we can predict that if the memory speeds are not increased in the same proportion for the two access modes, the bottleneck may bounce back and forth between the two buses.

With a 32bit bus system, let the ratio of random access cycle time to burst access cycle time be x so that $\mu_{random} = x\mu_{burst}$. To ensure that burst access would be our bottleneck for up to 20 processors, we could substitute the new processor and random access speed into

equations 4.6 and 4.7 to derive a proper value of x . Assuming that the processor speed is 4 times that of the random access cycle time, we have

$$\frac{10000}{\frac{5}{4}x\mu_{burst}(474 + 24.42 * 20)} \geq \frac{3}{8} * \frac{4}{\mu_{burst}}$$

$$x \leq 5.5$$

In other words, if we cannot keep the random access fast enough so that the cycle time will be less than 5.5 times that of the burst access, the bottleneck will be shifted to the random access bus. When the bus width is increased, this value of x will change. This is because the burst access bandwidth will increase by a greater extent than the random access bandwidth. Using equations 4.8 and 4.9, we have

$$\frac{10000}{\frac{5}{4}x\mu_{burst}(437 + 24.42 * 20)} \geq \frac{3}{8} * \frac{8}{\mu_{burst}}$$

$$x \leq 2.9$$

The small value of x implies that it is a lot harder to keep the bottleneck on the faster bus. Therefore, using a wider bus would not be a solution by itself.

The simulation result reveals a weakness in the analytical model for computing ρ_{random} . In the analysis, we have to estimate the average number of attempts for acquiring a lock using the assumption that when the number of processors increases, the number of attempts will also increase. In this design, we have made the improvement in assuming that the value of α , which is the time between retries, is a fraction of the average lock holding time instead of a fixed value to get a better estimate of the number of retries. However, ρ_{random} is still off from the simulated value by quite a large margin because the number of attempts also depends on how much time each processor spends on accessing shared information. With a more complex transfer syntax conversion step, the processor would spend more time in local processing and spend a smaller amount of time accessing shared information when compared to a processor doing simpler transfer syntax conversions. Thus, even if there is a large number of processors, the average number of attempts needed to acquire a lock can still be low

when the transfer syntax conversion is complicated. Since our estimate of ρ_{random} has not included the effects of the transfer syntax complexity, the estimated value would only be a pessimistic approximation. This is not important if we could ensure that the burst access bus is the bottleneck. Otherwise, care must be taken in using the estimate.

Another difficult problem is that we must maintain the buffering capability of the host interface so that data will not be lost. In previous designs, packets are moved out the interface immediately when they arrived so that the interface is always ready to accept more data. With this new approach, if the input rate is higher than the output rate due to bursty traffic, data may stay in the interface for a long time and thus special care is needed to make sure that we have enough buffer space in the interface.

Chapter 5

Results and Discussions

In the last few chapters, we have looked at how processing throughput and resource utilizations are affected by the number of processors, their speed and the location of the packets and program code. Here we are going to look at some other measurements of the system design and effects of other parameters on the system performance.

5.1 Cost

Table 5.1 summarizes the throughput and major characteristics for the various designs proposed. The trend is that the higher the maximum throughput, the more hardware and software changes are required.

Design	Throughput (Mbits/s)	Processor Utilization	Cost
SSM	60.08	0.6	use VRAM for the shared memory
LIM	164.16	0.81	same as SSM + use DRAM for the local memory
SPR	176.96	0.88	use VRAM for both the shared and local memory. DMA devices are needed for each processor and special instructions needed to relocate the packet
IPR	227.20	0.82	same as SPR + big interface buffers in the host and network adapter
IPR	565.90	0.85	same as SPR except that fast VRAM and a 64bit bus (2 times wider than other designs) are used

Table 5.1: Summary of system characteristics

The biggest jump in throughput is obtained when local memory is added to the SSM design. The improvement for using VRAM in the SPR design is small but it allows us to benefit from the faster burst access bus. With the optimizations implemented in the IPR

design, the throughput increases by close to 40% when compared to that of the LIM design. The ability to use faster memory with the IPR design allows us to push the throughput up to more than 9 times that of the SSM system. The main design feature which results in the improvement of the processing throughput is the addition of local memory. The cost of the system increases with the kind and number of memory devices added. Since the price of memory in terms of cost per bit is expected to drop with the advent of VLSI technology, the system proposed should be realizable with an affordable price tag. The major cost of the systems proposed comes from the hardware as the software component is essentially the same for all designs. The choice of a particular design for implementation depends on the throughput requirement and the budget.

5.2 Memory speed

The limiting factor in all our designs is the memory speed. In our work, we have only used a single memory speed except for the last design. With a faster memory cycle time, more transactions can be carried out with the use of faster processors and a higher throughput can be obtained. As we have discussed in the last chapter, cycle time of random and burst accesses have to be reduced by the same proportion to ensure that the system characteristic be preserved. Techniques like interleaving and split transactions can be used to increase the speed of random access effectively to compensate for the higher increase in burst access speed than the random access speed that are being seen in newer devices [17]. If the memory speed is increased to 10ns for burst access, ρ_{burst} will rise to over 2400Mbits/s which will enable our designs to work in a gigabit network.

5.3 Bus speed

Increasing the memory speed implies that the bus speed has to be increased and that more processing power is required to make use of the extra bandwidth. These would pose two problems in the bus design in terms of the speed of the bus and number of loads on the bus.

Currently, common bus design is limited to a speed of around 100MHz [36]. Problems of stray inductance and capacitance will become serious at higher speeds and special care is needed to make the system work. One may even have to resort to optical links for speeds higher than 400MHz. The number of loads on a bus affects the speed of the bus. The higher the number of loads, the lower the speed will be because of the loading effects. A bus design is unlikely to be able to support more than 40 to 50 loads.

To extend our design to gigabit network rates, we would need to double the speed of the various components. This is achievable with the various optimizing techniques mentioned above. However, to build a system that could handle even higher network rates based on a bus based shared memory architecture would seem unlikely because of the physical limitations of the bus design. Changes have to be made to the protocol to simplify the syntax conversion required and to put in more parallelism in the definition so that there will not be so much shared memory traffic. Other possible alternatives include going for bus sufficient systems or resorting to more expensive interconnection architectures.

5.4 Multiple connections

Our analysis has been limited to situations where there is only one active connection. All our packets originate from the same connection and this worst case scenario results in a heavy contention for the state information in the shared memory. In a real system, there will likely be more than one active connection and we can expect to have less bus and memory contentions and thus a higher system throughput than the results shown here. Takeuchi [37] had performed some simulations in this area and showed that this is the case.

5.5 Packet lengths

From equation 2.3, the maximum processing throughput is directly proportional to the packet length. The packet length that we have used in this context refers to the length of the presentation layer PDU. Figure 5.1 shows that variations in ρ_{random} when the packet length

is changed for the IPR design. We can observe that if the packet size is reduced, the bottleneck

Figure 5.1: Maximum throughput variation with different packet lengths

will go back to the random access bus. On the other hand, if the packet size is increased, ρ_{burst} will become the ultimate limit and memory devices with higher burst access speed can be used without creating the problem of having the random access bus as the bottleneck. The problem of having the random access bus as the bottleneck is that the random access bus traffic is less linearly related to the system throughput than the burst access traffic, and thus the accuracy of the estimate is lower.

5.6 Other presentation layer operations

In our discussions, we have only concentrated on the transfer syntax conversion operations in the presentation layer. In fact, there are other computation intensive functions in the presentation layer like the encryption operations for data security and compression operations for reduced bandwidth requirement. By using a similar argument as with syntax conversion, our design should be able to handle these functions as well with minimal changes.

5.7 Future Work

We have presented a high level analysis of a protocol processing system using multiple processors. The next step is to perform some detailed designs and to construct some critical system enabling circuits. These circuits include an efficient FCFS bus arbitration circuit and memory management circuit for the shared memory. The possibility of using interleaving or split transaction schemes should also be investigated. Once the basic hardware architecture has been defined, software can be ported to the system from some parallel implementations like the one described in [15] and further testing can be carried out.

Chapter 6

Conclusions

In this thesis report, three multiprocessor based OSI protocol processing systems have been described. The systems are designed to minimize the utilization of shared resources using techniques similar to those used for cache prefetching. An incremental approach had been used in which each system is designed with some improvements over the previous one. The results indicated that our designs are capable of maintaining a higher resource utilization and giving a higher processing throughput than previous published designs. We have also shown that the OSI protocol stack is promising in a high performance heterogeneous computing environment using a gigabit network despite of the computation intensive operations.

In the LIM design, local memories are used to store program codes. With this architecture, the processing throughput was increased by closed to 3 times as compared to the SSM design proposed in [37]. It was found that the false sharing of data packets in the shared memory had resulted in a severe bus congestion problem on the random access bus and this congestion had restricted the system processing throughput to a low value. The SPR design dealt with this problem by moving packet data to the local memory of the processor doing the processing before the operations begin. Our result indicated that by using the faster burst access bus to relocate the packets, we had successfully reduced the random access bus usage and shifted the bottleneck to the faster burst access bus. The high number of data movements and lack of fault tolerancy of this design prompted further changes. The IPR design improved on the SPR design by reducing the number of data relocation operations with the use of a big interface buffer. This design resulted in a throughput closed to 4 times that of the SSM design and allowed the system to provide a graceful degradation in performance when some

of the processors fail. With the use of faster memory devices and wider buses, a processing throughput of 560MBits/s has been estimated. The processing throughput of this design is close to half of that required in a gigabit network and almost an order of magnitude higher than that of the SSM design.

An analytical method for estimating the processing throughput for such systems has also been developed. Although many simplifying assumptions have been made in the analysis, good estimates were still obtained. From both the analytical and simulation results, we have identified that bus speed, memory speed, synchronization, number of active connections, packet lengths and transfer syntax complexities are among the more important parameters that control the processing throughput.

A bus based architecture is found to be sufficient for handling a network rate of over 500MBits/s. With further optimizations, it is possible to extend it to a gigabit rate. If higher network rates or processing power are needed, physical limits of the bus design must be reconsidered to ensure that proper operations can be maintained.

Using fast processors alone without the matching main memory is not enough to provide the necessary processing power needed for protocol processing. The constant flow of packets through the system makes data caches ineffective in bridging the growing gap between processor and memory speed. Memory speed is a critical factor in the system performance.

An equal proportion of increase in random access and burst access speed in VRAMs is important to enable our designs to scale properly. Similar to the problem of processor and memory speed difference, if access speeds of both access modes are not changed together, the performance of the system will be limited by the slowest device. This would make it uneconomical to use higher speed components in other parts of the system.

The need to process packets in sequence reduces the amount of parallelism that can be incorporated into the system. Operations like reassembly are found to slow down the processing of received packets and this type of operations should be revised to enable a better use of the resources. As stated by Amdahl's Law, the maximum speedup of a parallel

machine is inversely proportional to the amount of serial processing required. Therefore, to design protocol processing systems to work with gigabit rates and beyond, a parallel protocol specification not using finite state machines should be used to minimize the synchronizations needed and to allow a higher degree of parallelism.

Bibliography

- [1] W.Bain and S.Ahuja, "Performance Analysis of High Speed Digital Busses", Proceedings of the 1981 Conference on Computer Architecture, Minneapolis, Minnesota, May, 1981, pp. 107-133.
- [2] D.Banks and M.Prudence, "A High-Performance Network Architecture for a PA-RISC Workstation", IEEE Journal on Selected Areas in Communications, Vol. 11, No. 2, February, 1993, pp. 191-202.
- [3] T.Braun and M.Zitterbart, "High Performance Internetworking Protocol", 15th Conference on Local Computer Networks, October, 1990, pp. 172-179.
- [4] D.Bursky, "Combination DRAM-SRAM Removes Secondary Caches", Electronic Design, Vol. 40, No. 2, January, 1992, pp. 39-44.
- [5] D.Bursky, "Integrated Cached DRAM lets Data Flow at 100MHz", Electronic Design, Vol. 40, No. 4, February, 1992, pp. 142-146.
- [6] D.Callahan, K.Kennedy and A.Porterfield, "Software Prefetching", SIGARCH Computer Architecture News, Vol. 19, No. 2, April, 1991, pp. 40-52.
- [7] C.Chen and A.Somani, "Effects of Cache Traffic on Shared Bus Multiprocessor Systems", Proceedings of the 1992 International Conference on Parallel Processing, August, 1992, pp. I-285-I-288.
- [8] D.Clark, V.Jacobson, J.Romkey, H.Salwen, "An Analysis of TCP Processing Overhead", IEEE Communications Magazine, June, 1989, pp. 23-29.
- [9] D.Clark and D.Tennehouse, "Architectural Considerations for a New Generation of Protocols", SIGCOMM 90 Communications Architecture and Protocols, Computer Communications Review, Vol. 20, No. 4, September, 1990, pp. 200-208.
- [10] B.Davie, "A Host-Network Interface Architecture for ATM", SIGCOMM 91 Communications Architectures and Protocols, Computer Communication Review, Vol. 21, No. 4, September, 1991, pp. 307-315.
- [11] P.Denning, "On modeling program behavior", Proceedings of the Spring Joint Computer Conference, vol 40, AFIPS Press, Arlington Va., 1972, pp. 937-944.
- [12] M.Flynn, "Some Computer Organizations and Their Effectiveness", IEEE Transaction on Computers, 1972, pp.948-960.
- [13] R.Freund, "Heterogeneous Processing", Computer Vol. 26 No. 6, June, 1993, pp. 13-17.

- [14] D.Giarizzo, M.Kaiserswerth, T.Wicki and R.Williamson, "High-Speed Parallel Protocol Implementation", Proceedings of the IFIP WG 6.1/6.4 International Workshop on Protocols for High-Speed Networks, Zurich, Switzerland, May 1989, North-Holland, pp. 165-180.
- [15] M.Goldberg, G.Neufeld, M.Ito, "A Parallel Approach to OSI Connection-Oriented Protocols", Proceedings of the 3rd International IFIP WG6.1/6.4 Workshop on Protocols for High-Speed Networks, Stockholm, May, 1992, pp. 225-240.
- [16] J.Hennessy and N.Jouppi, "Computer Technology and Architecture: An Evolving Interaction", Computer, September, 1991, pp. 18-29.
- [17] *IC Memory Data Book*, Hitachi America Ltd., January, 1990.
- [18] C.Huitema and A. Doghri, "Defining faster transfer syntaxes for the OSI Presentation Protocol", Computer Communication Review Vol. 19 No. 5, October, 1989, pp. 44-55.
- [19] V.Jacobson, "Congestion Avoidance and Control", Computer Communication Review, Vol. 18, No. 4, August, 1988, pp. 314-329.
- [20] R.Jain, "Congestion Control in Computer Networks: Issues and Trends", IEEE Network Magazine, May, 1990, pp. 24-30.
- [21] N.Jain, M.Schwartz and T.Bashkow, "Transport Protocol Processing at GBPS Rates", Communications Architectures and Protocols: Proceeding of the ACM-SIGCOMM 90, Philadelphia, PA, September, 1990, pp. 188-199.
- [22] E.Johnson, "Working Set Prefetching for Cache Memories", Computer Architecture News, Vol. 17, No. 6, December, 1989, pp. 137-141.
- [23] C.Joseph, "A Parallel Algorithm for ASN.1 Encoding/Decoding", Master's Thesis, Department of Electrical Engineering, University of British Columbia, September, 1992.
- [24] N.Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers", Proceedings of the 17th International Symposium in Computer Architecture, 1990, pp. 364-373.
- [25] H.Kanakia and D.Cherton, "The VMP Network Adapter Board (NAB): High-Performance Network Communication for Multiprocessors", Communication Architectures and Protocols: Proceedings of the ACM-SIGCOMM 88, Stanford, CA, August, 1988, pp. 165-174.
- [26] A.Klaiber and H.Levy, "An Architecture for Software-Controlled Data Prefetching", Proceedings of the 18th International Symposium on Computer Architecture, 1991, pp. 43-53.
- [27] L.Kleinrock, *Queueing Systems*, A Wiley-Interscience publication, 1975.

- [28] D.McCrackin and B.Szabados, "Horizontal demand prefetching: a novel approach to eliminating the jump problem", Canadian Journal on Electrical and Computer Engineering, Vol. 16, No. 3, July, 1991, pp. 118-124.
- [29] M.McCutcheon, M.Ito, G.Neufeld, "Interfacing a Multiprocessor Protocol Engine to an ATM Network", Proceedings of the 3rd Internal IFIP WG6.1/6.4 Workshop on Protocols for High-Speed Networks, Stockholm, May, 1992, pp. 151-166.
- [30] J.Nicoud, "Video RAMS: Structure and Applications", IEEE MICRO, Vol. 8, No. 1, February, 1988, pp. 8-27.
- [31] A.Pleszkum and E.Davidson, "Structured Memory Access Architecture", Proceedings of the 1983 International Conference on Parallel Processing, 1983, pp. 461-471.
- [32] M.Rose, *The Open Book: A Practical Perspective on OSI*, Prentice-Hall Inc., 1990.
- [33] E.Russell, *Building Simulation Models with SIMSCRIPT II.5*, CACI, 1983.
- [34] A.Smith, "Sequential Program Prefetching in Memory Hierarchies", Computer, December, 1978, pp. 7-21.
- [35] A.Smith, "Cache Memories", Computing Surveys, Vol. 14, No. 3, September, 1982, pp. 473-530.
- [36] H.Stone and J.Cocke, "Computer Architecture in the 1990s", Computer, Vol. 24, No. 9, September, 1991, pp. 30-38.
- [37] L.Takeuchi, "Study of OSI Protocol Processing Engine", Master's Thesis, Department of Electrical Engineering, University of British Columbia, July, 1991.
- [38] M.Terada, T.Yokoyama, T.Hirata and S.Matsui, "A High Speed Protocol Processor to Execute OSI", IEEE Infocom Vol. 2, Piscataway, NJ, USA, 1991, pp. 944-949.
- [39] M.Zitterbart, "High-Speed Protocol Implementations Based on a Multiprocessor-Architecture", Proceedings of the IFIP WG 6.1/WG 6.4 International Workshop on Protocols for High-Speed Networks, Zurich, Switzerland, May 1989, North-Holland, pp. 151-163.
- [40] M.Zitterbart, "Parallelism in Communication Subsystems", IBM Research Report, 1992.

Appendix A

Simulation Details

Simulation is used to evaluate the different designs because it allows us to try out different configurations and combinations of parameters easier than implementing the system. It is also needed to study system behavior which can not be derived from the analysis presented because of the many assumptions that have been made to make the analysis tractable. Our simulation involves simulating the hardware and software of the system. The simulation program is written in SimScript II.5 and run on Sun Sparc workstations.

A.1 Simulation model

SimScript constructs for process and resource are used to model the system components. A process is defined as an object and the sequence of actions it experiences throughout its life in the model. A process routine is used to describe the activity of a process. A resource is defined as a passive object which is required by the process objects. If the resource is not available when required, the process object is made to wait until the resource becomes available.

Processors in the system are simulated as processes while buses and memory systems are simulated as resources. Instruction executions are simulated by obtaining the memory resource and/or the bus depending on whether the instruction is in the cache, local or shared memory and waiting for a fixed amount of time corresponding to the processor speed. The number and frequency of memory accesses for processing a packet is based on static code counts as used in other studies [37][23]. Only the processing of data packets, acknowledgement packets, connection packets and disconnection packets are simulated. Segmentation and

reassembly of packets are only done in the transport layer for simplicity. Since we are only interested in the maximum processing throughput, error and flow controls are not simulated.

The structured chart in figure A.1 shows the structure of the simulation program

Figure A.1: Overview of the simulation program

for a processing element. Not shown in the diagram are the network and host interface adapters which insert new jobs into the active queue. Monitoring routines which collect system statistics and routines which perform the initialization and termination operations are also not included. Three input files are needed and are used for specifying the system configuration, host data and network data. One output file is produced giving the system statistics at regular intervals and at the end of the simulation.

A.2 Protocol processing data

The data used for simulating the software or the timing aspect are based on the static code count. The set of data used in [37] is employed because it represents an unoptimized implementation on a single processor system. The data link layer statistics is for the IEEE 802.2 logical link control type I. The network and transport layer statistics are for the Internet Protocol (IP) and Transmission Control Protocol (TCP) respectively from an implementation in C done for Unix4.3 BSD by the University of California, Berkeley. The session and presentation layer statistics are for the OSI Session and Presentation protocol respectively from the ISODE implementation done in C (version 6.0) by Marshal Rose. Table A.1 shows the number of times a PDU is accessed by different layers. In our analytical model, we have

Protocol Layer	receive	send	value used in analysis
Datalink	6	6	6
Network	14	16	16
Transport	62	20	62
Session	60	40	60
Presentation	30	30	30

Table A.1: Number of access to PDUs for the different layers

simply taken the worst case disregarding the difference in send and receive operations for simplicity. Therefore, we have assume that the number of PDU access is $84\frac{l}{p} + 90$ where l is the length of the packet and p is the network packet size. For large packets, multiple packets would be needed in the lower three layers and thus more processing required.

Table A.2 shows the number of state and queue access in the different layers. Assuming

Protocol Layer	receive	send	value used in analysis
Datalink	0	0	0
Network	22	20	22
Transport	65	90	90
Session	40	30	40
Presentation	40	30	40

Table A.2: Number of state access for the different layers

that we need around 16 access for acquiring a lock which includes access to the queue and actually getting the lock, and using the maximum value among send and receive operations, the value used in our analysis becomes $(112 + 16 + r_t)\frac{l}{p} + 80 + 2 * 16 + r_s + r_p$ where r'_i s are the number of lock retries for the transport, session and presentation layers respectively.

Table A.3 shows the number of instructions executed in different layers. The average

Protocol Layer	receive	send	connect	disconnect	ack	average
Datalink	45	40	-	-	-	42.5
Network	160	160	-	-	-	160
Transport	375	400	250	250	200	355
Session	450	400	400	400	-	422.5
Presentation	400	350	350	350	-	372.5

Table A.3: Number of instruction access for the different layers

values used in the analytical model are obtained by assuming that 10% of the traffic comes from acknowledgments, 5% comes from connection requests, 5% comes from disconnection requests and the rest from packets¹. The number of instructions used in the analysis are obtained from the average values and is equaled to $557.5\frac{l}{p} + 795$.

Transfer syntax conversion is assumed to take 5-20 instructions for encoding and 5-40 instructions for decoding. The number of instructions are based on the ASN.1 encoder/decoder implemented by Mike Sample of the computer science department at the University of British Columbia [37]. We have further assumed that only $\frac{1}{4}$ of the packet data is copied during the conversion and thus the number of bytes accessed (read and write) is $\frac{l}{2}$

The amount of register only instructions is assumed to be 75% based on an educated guess. The actual number would depend on the processor used and the software implementation which could vary a lot. The accuracy of this estimation is verified by comparing the simulated processing latency time using this assumption with the published measurements for a RISC multiprocessor protocol processing system described in [15]. The simulated results were found to have the same order of magnitude with the measurements reported.

¹half of which from packets sent out and half from packets received

A.3 Simulation parameters

In our simulations, we have to set a number of parameters to enable us to observe the maximum processing throughput and to optimize the simulation time. We have used both a normal and poisson distribution for the network and host workload with a mean of 10000 and a standard deviation of 5000. The normal distribution is used in most simulations unless otherwise specified. Other parameters used are described in the following sections.

A.3.1 Obtaining Peak Performance

One way of achieving peak performance is to keep sending packets into the system. However, data sent into a system use a certain amount of processing power even before actual processing starts. For instances, active job queue needs to be updated, burst access bus may be used for moving the packet to the shared memory and locks may be acquired. All these are important shared resources and we would like to reduce this kind of over usage which would affect the result. In the simulation, a feedback mechanism is used to make sure that the input data rate is always higher than the output data rate so that all processors can be kept busy and at the same time that not too much extra packets would be sent. This is achieved by setting the input rate to about 4Mbits/s higher than the output rate and that the number of outstanding job in the active queue to be just greater than 3 times the number of processors. Since we want to find the peak throughput, flow control and error control are not simulated as they would reduce the peak rate.

A.3.2 Selecting the content of the workload

In a real system, the traffic going through the protocol processing system will consist of varying proportions of network and host packets. In our simulations, a 50-50 load is used. This would give us a good estimation when the peer are communicating on a equal basis. Different proportions of network and host packets have been simulated for the SPR design with 10 processors using a Poisson interarrival rate and an exponential service time to model

a realistic system. The results are shown in table A.4 and plotted in figure A.2. It is observed

Host traffic %	Net traffic %	Throughput (Mbits/s)
0	100	78.16
20	80	82.72
40	60	81.04
60	40	81.28
80	20	81.28
100	0	77.52

Table A.4: Effects on Throughput with different workload

from figure A.2 that a higher throughput is obtained in the middle of the graph where the ratio of host to network traffic is closer to 1. This is because when all the traffic comes from the same source, the chance of getting a lock contention is a lot higher and it reduces the overall performance.

Figure A.2: Throughput variation with contents of workload

A.4 Analysis of accuracy

In our simulations, random numbers are used. For an accurate simulation, the results should not be dependent on the random numbers used. The following sections describe what we have done to ensure that the choice of random numbers would not affect our simulation result.

A.4.1 Random numbers

SimScript generates pseudo-random numbers based on the Lehmer technique. A starting seed is multiplied by a constant to produce a new seed and a sample. In SimScript, 10 different random number streams are provided by default. These random number streams are used in our simulations and each one of them is used for a different purpose in an attempt to keep the results reproducible so that changing one part would not affect other parts in some unknown ways when the usage of a random number stream is changed. Further information about the random number generators in SimScript can be found in [33].

A.4.2 Sensitivity to random numbers

The result obtained from a single simulation gives us an estimate of the true value. This estimated value is affected by the random number stream and also the simulated time. Ideally, the result should not be sensitive to the choice of the random seeds. Using a long simulated time will reduce the effects of the random numbers but a long simulated time implies a long simulation time.

To be statistically accurate, each simulation should be repeated a number of times with different random seeds. However, a typical simulation takes up to 5-10 hours on a Sun Sparc station 2 workstation and to repeat every single run 10 or more times is not feasible for this research. Instead, we have looked at the SSM design where the random seed is exercised the least and tried to determine its sensitivity to the random seeds. The set of tests using different number of processors for the simple shared memory design was repeated 10 times.

Different random number streams provided by SimScript are used in each run. The resulting throughputs and bus utilizations are shown in Tables A.5, A.6 and A.7.

Trial	Number of Processors				
	1	2	4	8	16
1	0.482	0.858	1.820	3.457	5.348
2	0.537	0.972	1.876	3.658	5.400
3	0.558	1.078	2.141	3.838	5.668
4	0.558	1.074	2.008	3.730	5.397
5	0.479	1.012	1.946	3.601	5.397
6	0.537	0.972	1.840	3.727	5.432
7	0.559	1.074	1.961	3.733	5.238
8	0.480	0.971	1.914	3.623	5.456
9	0.480	0.971	1.913	3.776	5.415
10	0.482	0.982	1.990	3.877	5.564
mean	0.515	0.996	1.941	3.702	5.432
s.d.	0.037	0.067	0.093	0.123	0.116
%	7	7	5	3	2

Table A.5: Standard Deviation of Peak Throughput(MBytes/s)

Trial	Number of Processors				
	1	2	4	8	16
1	0.115	0.197	0.368	0.658	0.971
2	0.114	0.200	0.366	0.662	0.975
3	0.114	0.207	0.383	0.673	0.982
4	0.116	0.204	0.378	0.667	0.972
5	0.113	0.203	0.372	0.662	0.974
6	0.114	0.200	0.364	0.661	0.978
7	0.115	0.204	0.377	0.667	0.974
8	0.113	0.199	0.368	0.660	0.975
9	0.113	0.199	0.366	0.664	0.975
10	0.114	0.200	0.375	0.678	0.977
mean	0.114	0.201	0.372	0.665	0.975
s.d.	0.0010	0.0031	0.0063	0.0062	0.0031
%	1	2	2	1	0.3

Table A.6: Standard Deviation of Random Access Bus Usage

The standard deviation for the throughput values is less than 10% for all configurations. As the number of processors increases, the standard deviation decreases because

Trial	Number of Processors				
	1	2	4	8	16
1	0.023	0.035	0.069	0.122	0.196
2	0.025	0.040	0.071	0.132	0.200
3	0.026	0.044	0.080	0.137	0.210
4	0.025	0.042	0.076	0.135	0.199
5	0.023	0.041	0.072	0.128	0.202
6	0.025	0.040	0.069	0.134	0.202
7	0.026	0.043	0.074	0.135	0.195
8	0.023	0.040	0.072	0.131	0.202
9	0.023	0.040	0.071	0.135	0.202
10	0.023	0.040	0.075	0.140	0.205
mean	0.024	0.041	0.073	0.133	0.201
s.d.	0.0013	0.0024	0.0034	0.0050	0.0043
%	5	6	5	4	2

Table A.7: Standard Deviation of Burst Access Bus Usage

with more processing power, more packets get through and thus more random numbers are consumed and the initial choice of the seed is not as important.

The standard deviation for the bus utilization shows a similar trend although the variation is even smaller. This is mainly because during the startup phase of the simulation, resources are consumed but data do not get out of the system until after the required processing latency time. Since the throughput is computed based on the total data output over the entire simulated time, the effect of the startup phase actually reduces its value. The throughput obtained would thus underestimate the real value. This throughput value could be adjusted by eliminating the first part of the simulation when computing the throughput. That approach is not taken as we would like to be not too optimistic in establishing the peak values.

A.4.3 Picking the simulated time

Another set of simulations are carried out to observe the effects of the simulated time. As discussed above, a longer simulated time will reduce the bias introduced in the startup phase and thus will give us a higher value for the throughput. A longer simulated time will also

reduce the error introduced by not accounting for the jobs still running in the processor when the simulated time limit has been reached. Table A.8 shows the simulation results.

The results indicate that as the simulated time increases, the throughput also increases while the bus utilization remains the same. By using a 0.5s simulated time, we are underestimating the throughput by only 10% from the observed mean. The random and burst access bus utilizations are within 2% from the observed mean. This pessimistic estimation is useful for establishing a lower bound for the maximum throughput and bus utilization. In all of our simulations, a 0.5s simulated time is used.

Simulated time(s)	Throughput (Mbytes/s)	Random Bus Usage	Burst Bus Usage
0.5	0.482	0.115	0.023
1	0.500	0.111	0.023
2	0.519	0.114	0.024
3	0.534	0.113	0.024
4	0.533	0.113	0.024
5	0.520	0.112	0.024
mean	0.515	0.113	0.0237
s.d.	0.020	0.0014	0.0005
%	4	1	2

Table A.8: Variation due to different simulated time

Appendix B

queueing model

In section 2.3, we discussed what queueing model would be suitable for the packets entering the protocol processing system. Here, we will use the classical queueing theory and determine which queueing model is better. For the following analysis, assume that the system has a Poisson input rate and exponential service time.

B.1 Multiple queue multiple server

A multiple queue multiple server system can be represented by a M/M/1 queue for each of the server. Assume that the input rate is equal to λ , then for each server, the input rate will be $\frac{\lambda}{m}$. From queueing theory [27], the average delay time for each packet is given by:

$$\begin{aligned} T_m &= \frac{1}{\mu - \frac{\lambda}{m}} \\ &= \frac{m}{m\mu - \lambda} \end{aligned} \tag{B.10}$$

B.2 Single queue multiple server

From standard queueing analysis of a M/M/m queue [27], the average delay time is given by:

$$\begin{aligned} T_s &= \frac{1}{\mu} + \frac{P_Q}{m\mu - \lambda} \\ &= \frac{m - \frac{\lambda}{\mu} + P_Q}{m\mu - \lambda} \end{aligned} \tag{B.11}$$

$$\text{where } P_Q = \frac{1}{1 + m!(\frac{\mu}{\lambda})^m(1 - \rho) \sum_{n=0}^{m-1} \frac{1}{n!}(\frac{\lambda}{\mu})^n} \text{ Erlang C function}$$

Figure B.1: A multiple queue multiple servers system

$$\text{and } \rho = \frac{\lambda}{m\mu}$$

B.3 Comparing single queue and multiple queue systems

Comparing equations B.10 and B.11, it can be deduced that the single queue system will have a smaller average delay time if

$$\frac{\lambda}{\mu} > P_Q$$

It can be proved by mathematical induction that T_m is always greater than T_s .

Proof:

$$\text{To prove that } P_Q = \frac{1}{1 + m! \left(\frac{\mu}{\lambda}\right)^m (1 - \rho) \sum_{n=0}^{m-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n} < \frac{\lambda}{\mu}$$

For $m=2$,

$$\begin{aligned} P_Q &= \frac{1}{1 + 2\left(\frac{\mu}{\lambda}\right)^2 \left(1 - \frac{\lambda}{2\mu}\right) \left(1 + \frac{\lambda}{\mu}\right)} \\ &= \frac{1}{1 + \frac{2\mu^2}{\lambda^2} \left(\frac{2\mu - \lambda}{2\mu}\right) \left(\frac{\mu + \lambda}{\mu}\right)} \end{aligned}$$

Figure B.2: A single queue multiple servers system

$$\begin{aligned}
 &= \frac{1}{1 + \frac{2(2\mu - \lambda)(\mu + \lambda)}{2\lambda^2}} \\
 &= \frac{2\lambda^2}{2\lambda^2 + 4\mu^2 - 2\mu\lambda + 4\mu\lambda - 2\lambda^2} \\
 &= \frac{2\lambda^2}{4\mu^2 + 2\mu\lambda} \\
 &= \frac{\lambda^2}{\mu(2\mu + \lambda)} \\
 &= \frac{\lambda}{\mu} \left(\frac{\lambda}{2\mu + \lambda} \right) \\
 &< \frac{\lambda}{\mu}
 \end{aligned}$$

Thus the statement is true when $m = 2$.

Assume that the statement is true for $m = k$, thus $P_Q = \frac{1}{1 + k!(\frac{\mu}{\lambda})^k(1 - \rho) \sum_{n=0}^{k-1} \frac{1}{n!} (\frac{\lambda}{\mu})^n} < \frac{\lambda}{\mu}$ is true.

When $m = k + 1$,

$$\begin{aligned}
 P_Q &= \frac{1}{1 + (k+1)!(\frac{\mu}{\lambda})^{k+1}(1 - \rho) \sum_{n=0}^k \frac{1}{n!} (\frac{\lambda}{\mu})^n} \\
 &= \frac{1}{1 + (k+1)(\frac{\mu}{\lambda})k!(\frac{\mu}{\lambda})^k(1 - \rho) [\sum_{n=0}^{k-1} \frac{1}{n!} (\frac{\lambda}{\mu})^n + \frac{1}{k!} (\frac{\lambda}{\mu})^k]}
 \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{1 + (k+1)\left(\frac{\mu}{\lambda}\right)k!\left(\frac{\mu}{\lambda}\right)^k(1-\rho) \sum_{n=0}^{k-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n + (k+1)\left(\frac{\mu}{\lambda}\right)k!\left(\frac{\mu}{\lambda}\right)^k \frac{1}{k!} \left(\frac{\lambda}{\mu}\right)^k} \\
&< \frac{1}{1 + (k+1)\left(\frac{\mu}{\lambda}\right)\left(\frac{\mu}{\lambda} - 1\right) + (k+1)\left(\frac{\mu}{\lambda}\right)k!\left(\frac{\mu}{\lambda}\right)^k(1-\rho) \frac{1}{k!} \left(\frac{\lambda}{\mu}\right)^k} \\
&\quad \text{from assumption } k!\left(\frac{\mu}{\lambda}\right)^k(1-\rho) \sum_{n=0}^{k-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n > \frac{\mu}{\lambda} - 1 \\
&= \frac{1}{1 + (k+1)\left(\frac{\mu}{\lambda}\right)\left(\frac{\mu}{\lambda} - 1\right) + (k+1)\left(\frac{\mu}{\lambda}\right)\left(1 - \frac{\lambda}{(k+1)\mu}\right)} \\
&= \frac{1}{1 + (k+1)\left(\frac{\mu}{\lambda}\right)\left(\frac{\mu}{\lambda} - 1 + 1 - \frac{\lambda}{(k+1)\mu}\right)} \\
&= \frac{1}{1 + (k+1)\left(\frac{\mu}{\lambda}\right)\left[\frac{(k+1)\mu^2 - \lambda^2}{(k+1)\mu\lambda}\right]} \\
&= \frac{1}{1 + \frac{(k+1)\mu^2 - \lambda^2}{\lambda^2}} \\
&= \frac{1}{(k+1)\left(\frac{\mu}{\lambda}\right)^2} \\
&= \left(\frac{1}{k+1}\right)\left(\frac{\lambda}{\mu}\right)^2 \\
&= \frac{\lambda}{(k+1)\mu} \left(\frac{\lambda}{\mu}\right) \\
\text{thus } P_Q &< \frac{\lambda}{\mu} \text{ when } m = k+1 \\
&\quad \text{since } \frac{\lambda}{(k+1)\mu} = \rho \leq 1
\end{aligned}$$

Since the statement is true when $m=2$ and the statement is true when $m=k+1$ assuming that $m=k$ is true, by the principal of mathematical induction, the statement is true for all integer $m \geq 2$. (Q.E.D.)

Appendix C

Simulation Results

The following tables contain the simulation results of the various designs discussed in the thesis. These data have been plotted in a number of ways throughout the thesis.

number of processors	Throughput (Mbits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	3.84	0.115	0.023	1.00	1.00
2	6.88	0.197	0.035	0.90	1.80
4	14.56	0.368	0.069	0.95	3.80
8	27.68	0.658	0.122	0.90	7.20
12	37.68	0.868	0.171	0.82	9.84
14	41.04	0.937	0.191	0.76	10.64
16	42.80	0.971	0.196	0.70	11.20
20	43.52	0.995	0.204	0.57	11.40

Table C.1: SSM design with 10mips processors

number of processors	Throughput (Mbits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	7.12	0.163	0.037	1.00	1.00
2	14.0	0.296	0.065	0.98	1.96
4	26.0	0.537	0.117	0.91	3.64
8	46.56	0.875	0.204	0.82	6.56
16	52.48	0.998	0.237	0.46	7.36
20	52.00	0.998	0.239	0.37	7.40

Table C.2: SSM design with 20mips processors

number of processors	Throughput (MBits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	12.48	0.250	0.056	1.00	1.00
2	23.44	0.448	0.105	0.94	1.88
4	44.88	0.765	0.195	0.90	3.60
8	60.08	0.985	0.259	0.60	4.80
16	57.60	0.998	0.260	0.29	4.64

Table C.3: SSM design with 40mips processors

number of processors	Throughput (MBits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	3.84	0.052	0.023	1.00	1.00
2	7.04	0.076	0.037	0.92	1.84
4	14.64	0.137	0.069	0.95	3.80
8	30.24	0.256	0.134	0.98	7.84
16	57.76	0.478	0.261	0.94	15.04
20	72.32	0.579	0.320	0.94	18.80
24	85.44	0.689	0.383	0.93	22.32
28	99.12	0.791	0.433	0.92	25.76
32	107.84	0.880	0.470	0.88	28.16
36	109.60	0.959	0.479	0.79	32.94

Table C.4: LIM design with 10mips processors

number of processors	Throughput (MBits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	7.12	0.067	0.037	1.00	1.00
2	14.56	0.115	0.067	1.00	2.00
4	28.88	0.209	0.128	1.00	4.00
8	56.64	0.388	0.245	0.99	7.92
16	109.28	0.710	0.460	0.96	15.36
20	130.48	0.845	0.554	0.92	18.40
24	143.12	0.939	0.602	0.84	20.16
28	135.20	0.988	0.585	0.68	19.04

Table C.5: LIM design with 20mips processors

number of processors	Throughput (Mbits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	12.72	0.097	0.060	1.00	1.00
2	25.12	0.173	0.112	0.99	1.98
4	50.40	0.318	0.216	0.99	3.96
8	99.12	0.587	0.415	0.97	7.76
12	140.64	0.805	0.581	0.92	11.04
14	153.60	0.876	0.631	0.86	12.04
16	164.16	0.939	0.667	0.81	12.96
20	164.72	0.992	0.673	0.65	13.00

Table C.6: LIM design with 40mips processors

number of processors	Throughput (Mbits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	3.84	0.031	0.029	1.00	1.00
2	6.88	0.039	0.046	0.90	1.80
4	14.64	0.062	0.093	0.95	3.80
8	30.88	0.111	0.190	1.00	8.00
16	60.56	0.201	0.379	0.99	15.84
20	74.96	0.248	0.473	0.98	19.60
32	116.48	0.395	0.730	0.95	30.40
40	138.00	0.508	0.882	0.90	36.00
44	154.40	0.558	0.946	0.91	40.04
45	150.56	0.618	0.906	0.87	39.15
48	108.72	0.737	0.744	0.59	28.32

Table C.7: SPR design with 10mips processors

number of processors	Throughput (Mbits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	7.28	0.037	0.047	1.00	1.00
2	14.32	0.054	0.088	0.98	1.96
4	28.16	0.089	0.167	0.97	3.88
8	56.40	0.161	0.336	0.97	7.76
16	113.92	0.313	0.675	0.98	15.68
20	136.64	0.376	0.806	0.94	18.80
24	157.44	0.439	0.914	0.90	21.60
26	165.20	0.478	0.967	0.87	22.62
27	169.92	0.489	0.988	0.86	23.22
28	176.80	0.515	0.999	0.87	24.36

Table C.8: SPR design with 20mips processors

number of processors	Throughput (MBits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	12.64	0.047	0.077	1.00	1.00
2	25.12	0.075	0.148	0.99	1.98
4	50.24	0.132	0.289	0.99	3.96
8	100.32	0.248	0.575	0.99	7.92
12	146.88	0.358	0.830	0.97	11.64
14	162.56	0.397	0.908	0.92	12.88
15	169.52	0.418	0.945	0.89	13.35
16	176.96	0.438	0.981	0.88	14.08

Table C.9: SPR design with 40mips processors

number of processors	Throughput (MBits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	13.92	0.048	0.066	1.00	1.00
2	28.40	0.080	0.135	1.00	2.00
4	52.96	0.133	0.256	0.95	3.80
8	104.56	0.246	0.494	0.94	7.52
16	195.28	0.448	0.873	0.88	14.08
20	227.20	0.520	0.986	0.82	16.40

Table C.10: IPR design using 40mips processors

number of processors	Throughput (MBits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	7.12	0.034	0.034	1.00	1.00
2	14.80	0.050	0.071	1.00	2.00
4	30.48	0.085	0.148	1.00	4.00
8	56.73	0.143	0.276	1.00	8.00
16	114.21	0.273	0.542	1.00	16.00
20	137.31	0.325	0.648	0.96	19.20
32	210.82	0.493	0.937	0.93	29.76
36	228.86	0.543	0.987	0.89	32.04

Table C.11: IPR design using 40mips processors and 2 times transfer syntax operations

number of processors	Throughput (Mbits/s)	Random access bus utilization	Burst access bus utilization	Processor utilization	Actual speedup
1	16.63	0.038	0.025	1.00	1.00
2	33.43	0.058	0.050	1.00	2.00
4	64.41	0.096	0.096	0.97	3.88
8	133.94	0.184	0.198	1.00	8.00
16	262.02	0.341	0.382	0.98	15.68
20	326.34	0.417	0.469	0.98	19.60
26	409.67	0.530	0.582	0.95	24.70
32	488.83	0.640	0.685	0.92	29.44
40	565.90	0.772	0.777	0.85	34.00
50	547.95	0.881	0.804	0.66	33.00

Table C.12: IPR using 40mips processors and fast memory

Appendix D

Glossary of terms

ASN	Abstract syntax notation
BAB	Burst access bus
DMA	Direct memory access
DRAM	Dynamic random access memory
FCFS	First come first serve
IPR	Improved packet relocation
LIM	Local instruction memory
MIMD	Multiple instruction multiple data
MIPS	Million instructions per second
OSI	Open system interconnection
PCI	Protocol control information
PDU	Protocol data unit
RAB	Random access bus
RISC	Reduced instruction set computer
SAP	Service access point
SDU	Service data unit
SIMD	Single instruction multiple data
SPR	Simple packet relocation
SSM	Simple shared memory
VRAM	Video random access memory