SCS&E Report 9406
March, 1994

A Parallel Approach to High-Speed Protocol Processing

Toong Shoon Chan and Ian Gorton

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
THE UNIVERSITY OF NEW SOUTH WALES

Abstract

A rapid increase in the transmission bandwidth of optical networks has created a bottleneck in protocol processing at the end systems. This has resulted in the inability of applications and network protocols to exploit the full bandwidth of a high-speed network. This paper presents a parallel architecture that is designed to support high-speed protocol processing. The advent of the T9000 transputer and C104 router technology has provided a platform that is suitable for the construction of a highly parallel and scalable protocol processing architecture based on packet and functional parallelism. A simulation of the architecture has been implemented and has demonstrated the advantage of exploiting a parallel architecture for protocol processing.

E-mail:
chants, iango@cse.unsw.edu.au

1. INTRODUCTION

Technological advancements in transmission speeds of optical fibre media have resulted in much research and development into high-speed networks. More recently, effort has been directed into developing a high-speed wide area network known as the broadband integrated services digital network (B-ISDN) that is able to provide a diverse range of services, ranging from data communication to transportation of video data. Although the processing speed of communication processors has risen steadily over the past years, they are still not able to match the transmission speed of optical media. This has resulted in the inability of applications and network protocols to exploit the full bandwidth of a high-speed network.

Consequently, several approaches to solve this communication bottleneck have been proposed. The first approach concerns with the optimal implementation of existing protocols, such as TCP and TP4. It has been shown in [1] that efficient implementations of TCP for Berkeley BSD Unix can still achieve a relatively high throughput performance. Although this is a feasible approach, careful examination reveals that several mechanisms employed in TCP to ensure reliable transport of data are not suitable for high-speed networks. This is because TCP was designed in the era when bandwidth was expensive and scarce. This has resulted in high overheads, as such protocols were designed to be robust in the face of adverse conditions. High protocol overheads were necessary for the proper control of the transmission of data, so that better utilisation of expensive bandwidth could be achieved. In view of the abundant availability of bandwidth and greatly improved characteristics of optical networks, such overheads are unnecessary and further burden the communication processing.

The next approach concerns new protocol designs that are suitable for high-speed operation. The key to such design is to minimise protocol processing requirements and consequently improve protocol processing speed. This has led to design of several lightweight protocols [2, 3] that are designed to provide high-speed processing by simplifying the protocol operation under normal data transfer phase, without compromising on the functionality to provide reliability of data transfer. Often, these lightweight protocols are designed for ease of implementation in hardware to further speed up in protocol processing.

The next approach is concerned with the implementation of protocols on an outboard hardware protocol processor platform. Such implementations can come in the form of dedicated VLSI hardware for a specific protocol [4] or using a general multi-processor platform. The main advantage of a dedicated VLSI hardware solution is the high processing speed achievable and consequently higher throughput performance. However, this advantage is offset by the high cost and difficulty in adapting to changing requirements. On the other hand, implementations using general multi-processor platform, like the transputer, offer greater flexibility and are achievable at modest cost. Importantly, the scalability of transputer networks mean that additional processors can be added in future if needs arise. This is a desirable feature, since it is anticipated that transmission speed of optical media will continue to increase into multi-giga bits range.

In this paper, we will focus on applying parallelism to both transport and presentation layers based on a message passing parallel architecture. Section 2 highlights the different levels of parallelism possible for protocol processing. Section 3 discusses

the protocol processing requirements. The suitability of transputer for protocol processing is discussed in section 4. Section 5 presents a highly parallel architecture for protocol processing using the T9000 and C104 router. In section 6, a simulation of the proposed architecture is presented and the performance results are presented in section 7. Finally, section 8 describes the future work and conclusions.

2. LEVELS OF PARALLELISM IN PROTOCOL PROCESSING

This section briefly describes the options that are available for parallel implementation of protocol processing. The levels of parallelism identified are as follows:

i. Connection Parallelism

The highest level of parallelism that can be applied to protocol processing design deals with connection parallelism. In order to establish a communication link between end systems, a connection has to be set up. Parallel processing can be applied to the established connections by dedicating each active connection to a separate processor. This can be achieved easily since each of the active connections operates almost independently from each other. After a connection is established, the application is mapped to the assigned protocol processor, to which all subsequent packets associated with that connection will be routed. The advantage of this parallel architecture is the ease of implementation, with minimal synchronisation required between the parallel connections. However, this architecture does not perform well from the point of processor utilisation, as the processor utilisation factor is dependent on the number of connections established, and the utilisation factor for each individual connection. One solution is to implement a dynamically reconfigurable multi-processor network to optimise the processor utilisation factor. Although feasible, this could make the synchronisation between processors complex, thus introducing unnecessarily high overheads.

ii. Pipeline Parallelism

A protocol architecture based on the layering principle can exploit pipeline parallelism. It is well known that layering in communication architecture reduces the design complexity by organising various protocol services into separate layers. Each of these layers operates almost independently, while providing services to higher layers. Pipeline parallelism is concerned with assigning these layers to separate processors. As a packet arrives, the dedicated processor performs the necessary processing on the packet for that layer and passes the processed packet to the next higher layer. The advantage of such parallelism is the ease of implementation, since the layering principle adapts naturally to pipeline parallelism. As with any pipeline parallelism, the overall throughput achievable for an implementation is limited by the slowest layer processor. Therefore, if a particular layer is found to be exceptionally slow (like the presentation layer), it is important to optimise this layer to ensure overall

performance gain. One possible solution is to apply functional parallelism on the bottleneck layer to further speed up the processing.

iii. Functional Parallelism

Early protocols (eg. TCP) employ in-band signalling for the exchange of information between peer protocol layer entities. With in-band signalling, the control and data functions are combined together as a single entity. At the receiving end, each packet is parsed to determine the control information that is present, so as to invoke the necessary actions to ensure correct operation. This means that the processing of control information for each received packet is inherently sequential, and this leads to increased processing time for the embedded data portion. As a result, performing functional parallelism on these protocols requires high overheads in synchronising between functional units. On the contrary, out-of-band signalling aims to transmit the control information and data as separate packets. Therefore, at the receiving end, a simple hardware circuit can be built to identify different packets, which can then be routed to the corresponding processing units. The major advantage of out-of-band signalling is the ability to support functional parallelism efficiently. Therefore, the entire protocol processing can be separated into several stand-alone functional units, with minimum interaction among the functions. The drawback of out-of-band signalling is the increased bandwidth needed for transmission of individual control packets. However, with a quantum increase in available bandwidth in high-speed communication systems and the ability to support efficient functional parallel architectures, the out-of-band signalling technique is an attractive proposition for incorporation in future protocol designs.

iv. Packet Parallelism

The next level of parallelism in protocol processing deals with packet parallelism. Each incoming data packet is assigned to one processor from a pool of data packet processors, while incoming control packets are assigned to a dedicated control packet processor. Implementing such parallelism allows several data packets to be processed in parallel, thereby achieving high throughput performance. Such an architecture is particularly attractive if serial processing per data packet is highly complex, resulting in high concurrency, while minimising the computation cost of synchronisation between processors. Importantly, packet parallelism is highly scalable, where additional data packet processors can be added in future, if the need arises. This is in contrast with functional parallelism, where parallelism is limited by the ability to isolate functions, such that each function is essentially independent from each other.

Each option presented here can potentially increase the overall throughput performance of protocol processing. In order to achieve maximum performance gain, it is important to apply parallelism such that processor utilisation is maximised, while minimising synchronisation requirements associated with parallel processing. In this

paper, we will focus on applying both functional and packet parallelism to protocol processing at transport and presentation layers, which are seen to be the processing bottleneck[5].

3. PROCESSING REQUIREMENTS

3.1 Data and Control Processing

Despite the complexity of protocol processing, the entire protocol operation can generally be grouped into two processing functions: data processing and control processing. Data processing is concerned with the transfer of data between communicating end points. Typical operations may include movement of data from memory to memory, checksumming of data packets, data packet formatting and sequencing of data packets. On the other hand, control processing is concerned with the management of communicating connections and regulating of data transfer. Operations include connection establishment and close, regulating data transfer by exchanging of control/synchronisation information and monitoring of network connections. Generally, a control packet would require a relatively low number of bytes to convey its control information (of the order of 100 bytes). At the receiving end, the various control fields are parsed and the necessary control actions are generated. In contrast, data processing would require much higher computation with typical packet size in the excess of 2000 bytes. Any access or formatting on this data at any protocol layer will result in high computational overheads. At the transport layer, a typical data access operation is to compute the checksum for error detection. At the presentation layer, formatting is required to convert the data to/from an abstract syntax representation. An example of such a standard representation is the ISO ASN.1. If it requires one instruction per byte to convert a data packet of 2000 bytes in size, this would require a total of 2000 instructions to complete each packet.

3.2 Out-of-band Processing

As mentioned in the previous section, early protocols (eg. TCP) tend to multiplex both control and data information onto a single packet. This has resulted in the difficulty of applying functional parallelism to control and data processing. In addition, separating the processing of control and data information using out-of-band signalling enables the end points to synchronise less frequently. In view of the much improved error characteristics of future networks and the complexity in control processing, it is desirable that such processing be less frequent. Further, avoiding the generation of control information on per packet basis[1] reduces the frequency of synchronisation between data and control processing at the receiving end. This results in higher concurrency in terms of packet parallelism, since synchronisation between packet and control processors is less frequent.

---

[1]In TCP, an acknowledgment is generated for each received data packet.

3.3 Data Ordering Constraints

In order to apply packet parallelism to protocol processing, we need to employ the concept of Application Level Framing (ALF) [6]. In the OSI protocol layering, each protocol layer appends its own protocol header onto the front of the data unit received from the preceding layer. At the transport layer, data units from the session layer might be segmented into smaller data units suitable for transmission to the network. At the receiving end, these segmented data units needs to be reassembled in the correct order by the peer transport layer, before being processed by upper layers. This segmentation of data units at the transport layer results in the inability to apply packet parallelism to protocol processing for higher layers, since data ordering is required. In contrast, ALF uses a single autonomous data unit for all the processing functions required at different layers. The data unit to be transferred is based on the application data unit, such that each unit is properly framed and can be processed independently by upper layers to achieve high concurrency. Segmentation and reassembly is performed at the network layer, where transport data units are segmented into smaller data units[2] suitable for transfer over the network.

4. THE TRANSPUTER FOR PROTOCOL PROCESSING

A number of features of Occam and the transputer help make a protocol implementation efficient both in terms of programming effort and execution speed. This section highlights several desirable characteristics for protocol processing that are inherent in the design of transputer and Occam.

i. Programmability

Although it is feasible to implement a protocol on specialised hardwired VLSI to achieve high processing speed, this approach often results in difficulty in adapting to changing protocol specification. The ability of a protocol processor to be programmable is essential, since it is envisaged that a protocol specification is unlikely to be stable from the outset. Instead, a protocol specification can continue to evolve as a result of further development. For example, although TCP has been around for more than a decade, it has continued to evolve due to changing requirements. In contrast to hardwired VLSI implementations, the transputer is a general purpose programmable microprocessor that is designed to support efficient parallel processing. Occam is the programming model for transputers. As such, all transputer instruction sets are designed to provide highly efficient compilation of programs written in Occam.

---

[2] In B-ISDN, higher level protocol data units are segmented into ATM cells of 53 bytes.

ii. Instruction Primitives

Although not designed specifically for protocol implementation, a number of instruction primitives available on the transputer greatly simplifies the programming effort and consequently results in efficient implementation.

In computer communication, timers are often used extensively in protocols for the protection against loss of messages by monitoring the elapsed time associated with messages. Further, timers are also used to detect failures in communicating connections. On a processor that does not directly support the implementation of timers, the computational overheads in managing these timers can prove to be very high [7]. In contrast, the support in Occam for timers greatly simplifies the implementation of timing calculations and consequently results in higher computational efficiency.

To protect against the corruption of data messages, cyclic redundancy checking (CRC) is often employed to check on the data integrity. The provision of transputer to directly support byte or word CRC computation through CRC primitive instructions results in efficient implementation. These instructions can be used on arbitrary length serial data streams.

One way of interfacing network peripherals to the transputer is to use external links to transfer packets from the network to the processor. The support in Occam for definition of channel protocols allows fields to be extracted from the received packet almost instantly, without the need to parse the entire packet. Similarly, complex protocol structures can be constructed and transferred to the network peripheral using the same method. The provision of such primitives results in the ease of protocol definition and programming, while optimising protocol execution.

iii. Fast Context Switching

Protocol processors are required to perform processing on several connections and across multiple protocol layers. Therefore, it is important that protocol processor provides a fast context switching mechanism to minimise the overheads of switching between connections and protocol layers. The incorporation of a hardware scheduler within the transputer and efficient context switching architecture results in a sub-microsecond interrupt response time.

iv. Multiple Links Architecture

The transputer is designed with four links for interfacing to external peripherals, as well as communicating between transputers. The provision of multiple links supports a high degree of concurrency on the transputer by providing multi-port access to the processor. For example, one link can be interfaced to the network to provide data input, while another link is interfaced to the host to provide data output. Together with the DMA capability of each link, concurrent data transfer without involvement of the processor is possible.

v. Scalability

It is anticipated that the bandwidth of future networks will continue to increase, thus, resulting in increased protocol processing requirements at the hosts. The advent of T9000 and C104 router technology could provide a platform necessary for the construction of a highly parallel and scalable protocol processing system. Therefore, if the need arises, additional processors can be added to match the required processing capacity.

5. ARCHITECTURE OVERVIEW

This section presents a highly parallel protocol processing architecture that is based on functional and packet parallelism. The architecture is designed to support high-speed protocol processing for transport and presentation layers using the T9000 transputer. An overview of the architecture is shown in Figure 1. All transputers are interconnected via a C104 router chip to allow processes distributed on up to 32 transputers to communicate. The data processors are responsible for the processing of data packets, with each processor comprising local memory for storage of incoming data packets. The control processor is responsible for the management of connections and regulating data transfer.

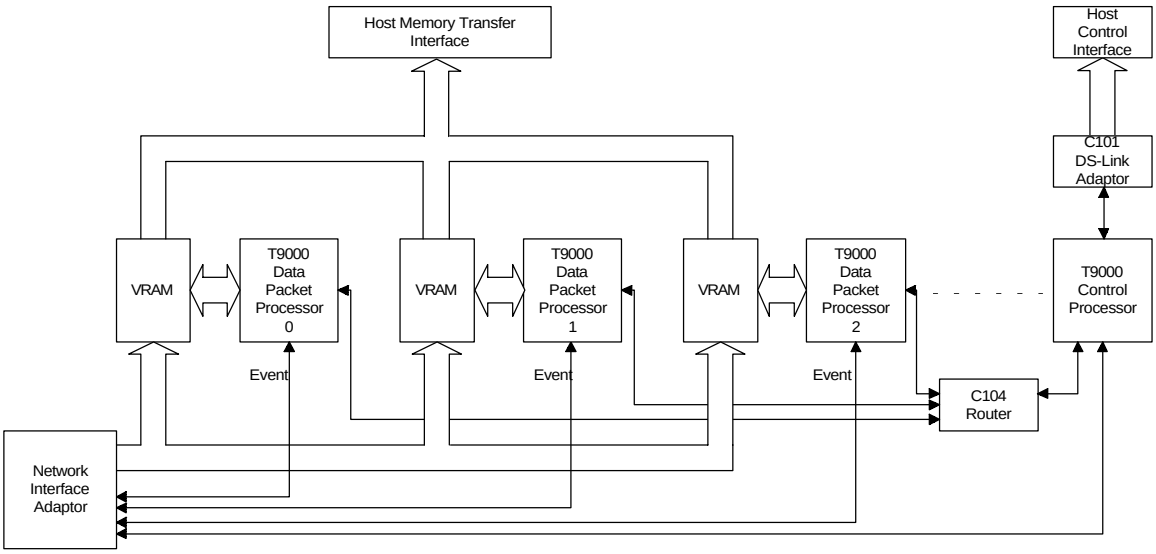

Figure 1 Architecture Overview

5.1 Packet Scheduling

Data packets arriving from the network are assigned to one of the data packet processors in a deterministic fashion, based on the result of a modulo division of the sequence number for the packet over the total number of data packet processors. For example, if the incoming data packet contains a sequence number of 45 and the total number of data packet processors is 10, based on the result of the modular division of 45

over 10, the packet is assigned to data packet processor 5. While it is possible to copy the data packet into the local memory of the assigned processor via the transputer serial link, the large data packet size to be transferred will impose unnecessary overheads in the copying process. Therefore, a dual-ported approach based on conventional dual-port video RAM (VRAM) is employed. One of the advantages of using VRAMs over random access dual-port static RAMs is its cost effectiveness. More importantly, the interfacing between the serial access port of VRAMs and the network adapter requires minimal arbitration logic, while enabling high-speed serial transfer rate of up to 33 MHz. For example, consider a serial transfer width of 32 bits, a data transfer rate in the excess of 1G bps is achievable. Once the incoming data packet has been transferred to the serial access port of the respective VRAM, the network adapter signals the assigned processor via one of the event interrupt pins, before transferring the data packet into the random access memory of the VRAM for processing.

5.2 Packet Synchronisation

Packet synchronisation is required for coordinating among the data packet processors to ensure data packet ordering. As mentioned, data packets are assigned to processors in a deterministic fashion, with each processor managing its own data ordering. For example, if the system consists of 10 data packet processors, then for processor 0, the expected data packet sequence ordering is 0, 10, 20, 30,...., while processor 1's expected data packet sequence ordering is 1, 11, 21, 31,.... and so on. In short, each processor is responsible for detection of its local missing, duplicate or out-of-order packets. To synchronise the overall ordering of packets among the processors, a mechanism known as *token sequencing* has been devised. When a connection is established, a token is created on the processor that is expected to receive the first data packet. In addition, for each connection, a table containing two fields <*seq*, *presentation_ok*> is created on each data packet processor, where *seq* contains the sequence number received and *presentation_ok* indicates if the corresponding packet has been processed by the presentation layer. The basic processing model on each data packet processor is shown in Figure 2. On the receipt of each data packet, the transport layer will check on the validity of the packet and the sequence number information is updated in the table. Subsequently, the address of the packet is passed to the presentation layer for processing of the data. Once the presentation layer has completed processing the packet, it will signal the transport layer and the *presentation_ok* field for that corresponding packet is updated. If the entry on the top of the table has a sequence number equal to the token value and the presentation processing on the data has been completed, the processed data is transferred to the host. The token value is then incremented and passed on to the next neighbour processor in sequence. Thus, the simple token sequencing mechanism ensures the correct ordering of data is presented to the host, while requiring minimum overhead in synchronising between processors.

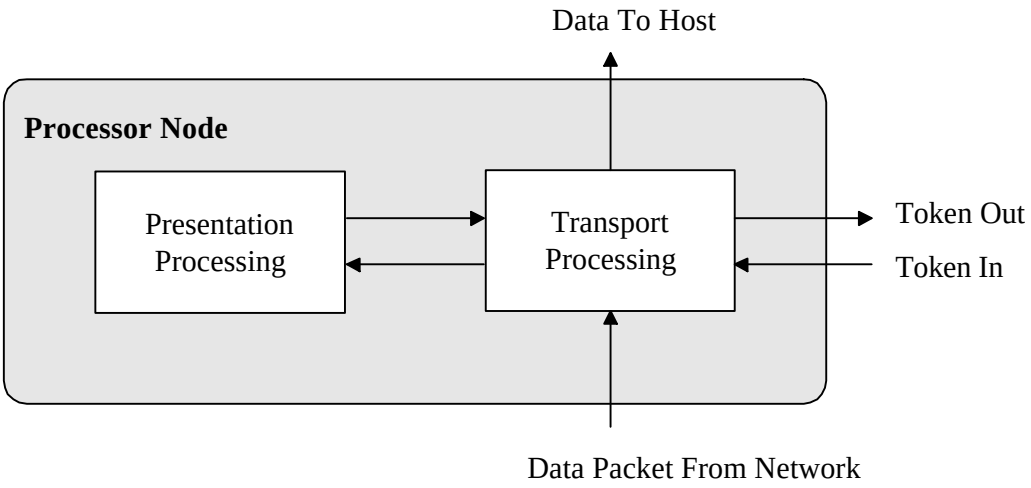

Figure 2 Basic Processing Model At Processor Node

5.3 State Synchronisation

A high-performance transport protocol, HTPNET [8], has been simulated on the proposed architecture. To support functional parallelism, HTPNET employs out-of-band signalling, in which peer-transport entities exchange data and state information using separate packets. A state information packet is exchanged frequently so that the state between the transmitter and receiver is synchronised. The basic state synchronisation mechanism is shown in Figure 3. The state information exchange is initiated periodically by the transmitter. Each state packet sent from the transmitter is associated with a synchronisation value that is incremented after each transmission. Similarly, data packets that are sent before the next state exchange are associated with the next synchronisation value in sequence. For example, if the next state exchange will contain a synchronisation value of 20, then those data packets that are sent between synchronisation value 19 and 20 will be associated with synchronisation value 20. Periodically, a state packet is admitted into the network which traverses the same path as the preceding data packets. When the state packet arrives at the receiver, it is transferred to the control processor. The control processor then initiates a state synchronisation with all the data packet processors to request the complete state and error control information from each data packet processor. Upon receipt of the state information from all the data packet processors, the control processor assembles the information and also copies the received synchronisation value to construct an outgoing state packet. Upon receipt of the return state packet, the transmitter control processor updates the status of the transmitted data packets by comparing the synchronisation values of the transmitted data packets with the echoed synchronisation value. Consequently, for unacknowledged data packets whose associated synchronisation value is less than or equal to the echoed synchronisation value, a retransmission is necessary.

For each received state packet at the transmitter control processor, a bit-map containing the status of the data packets is assembled and communicated to the data packet processors, which perform the necessary retransmission and release of data packet

memory.

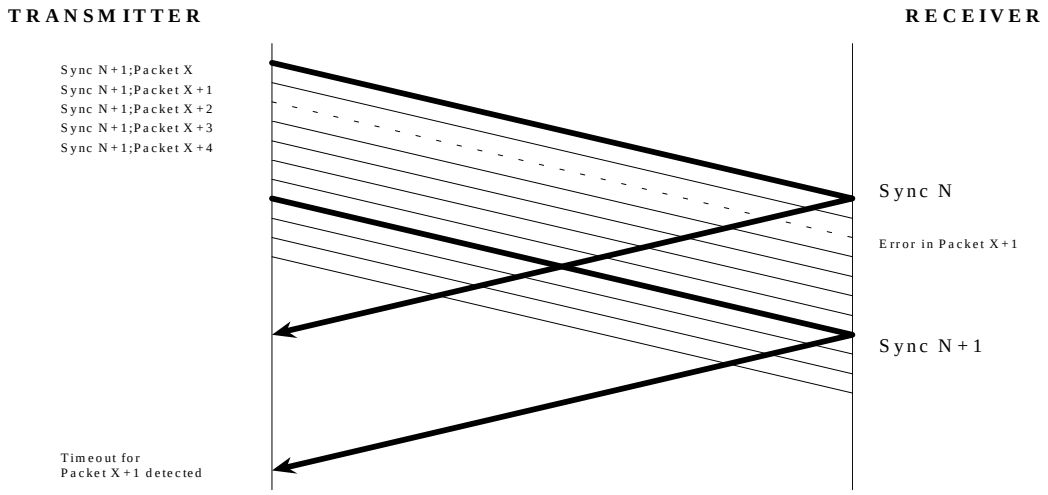

Figure 3 Basic State Synchronisation Operation

6. Simulation

A simulation of the proposed system has been implemented to evaluate the performance of the architecture. A simulation tool known as NETSIM [9] has been used for the simulation of the proposed system. The simulation program is written with the T9000 as the target processor. Before performing the actual simulation, the processes to be simulated on the proposed architecture were written in Occam. These processes were compiled and tested on a single T800 transputer to verify the functionality of the program. Although, it is possible to test on a single T800 transputer, the architecture cannot be extended to run on multiple T800 transputers due to the limitation on the number of external links available. However, this limitation has been eliminated with the T9000 transputer and C104 router [10, 11]. After verifying the functionality of the program, the Occam processes were translated to equivalent C functions for execution in NETSIM. The C104 router is not simulated, instead, links between processors are connected directly. Omitting the simulation of C104 router should have minimal effect on the overall performance measurement of the system, since messages passed between the processors are short and infrequent. In addition, the high operating speed of the C104 and the DMA capability of the transputer links further reduces the overhead of passing massages between processors.

To simulate multiple processes executing on a single processor, a resource facility employing a round-robin discipline is simulated. The simulated processor will serve only one process at a time, while suspended processes are queued. The time slice period is set to 5120 cycles[3] of the external clock. A process is permitted to run until it has completed its action or is descheduled whilst waiting for communication from another process. Modelling the advancement of simulation time while executing the instructions is achieved by identifying the basic block within each process and applying the necessary delay to

[3]The choice of a timeslice period of 5120 cycles is based on the design of T800 transputer [13].

execute these blocks [12]. A basic block is defined as a sequence of instructions such that all instructions within that block are executed in sequence until a process interaction point is reached. Some examples of a basic block are shown in Figure 4. After executing each basic block, the process is scheduled to delay for a time proportional to the number of machine cycles required to execute on the target processor[4]. The number of instructions required to perform presentation processing is estimated to be about 10 instructions per byte. The serial access speed of VRAM is set to operate at 33 MHz, with a access bus width of 32 bits.

```
-- Basic Block Starts
SEQ
  -- Sequence of Instructions
  ....
  ....
  -- Process Interaction
  a ! b
  -- Basic Block Ends
```

```
ALT
  -- Basic Block Starts
  a ? b
    SEQ
      -- Sequence of Instructions
      ....
      ....
      -- Basic Block Ends
  c ? d
    SEQ
      ....
```

Figure 4 Examples of Basic Block

7. PERFORMANCE RESULTS

In order to fully evaluate the performance of the system, various configurations of the architecture are simulated. To allow ease of reconfiguring the system, processes in the architecture are written to be generic in nature, such that the software remains unchanged even with different processors configuration, thus making the system inherently scalable.

Figure 5 shows the performance of the system with processor speed set to 20 MHz and network speed of 200 Mbps. The throughput increases almost linearly under low processor configuration, while the processor's utilisation remains almost constant at 100%. The throughput continues to increase linearly until about 12 processors, when the processing capacity of the system is beginning to match the network capacity. Meanwhile, the processor utilisation begins to decline with increasing number of processors. Figure 6 shows the performance of the system with processor speed of 50 MHz, while network speed remains at 200 Mbps. As expected, due to the faster processor speed, the number of processors required to match the network speed is reduced to about 6 processors.

[4]The number of machine cycles required to execute Occam programs is based on the performance figure obtained from [14].

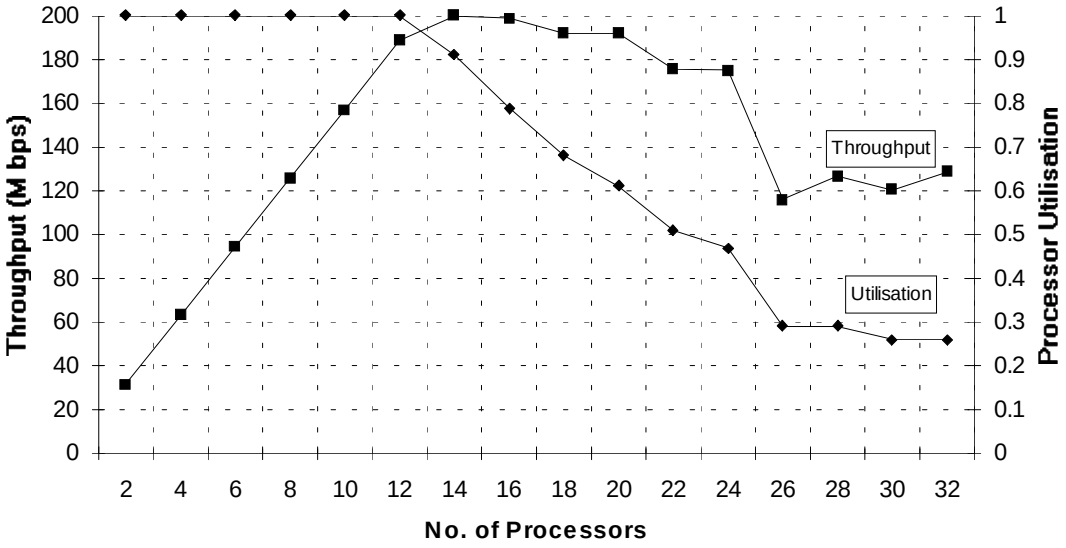

Figure 5 Performance With Processor Speed of 20 MHz

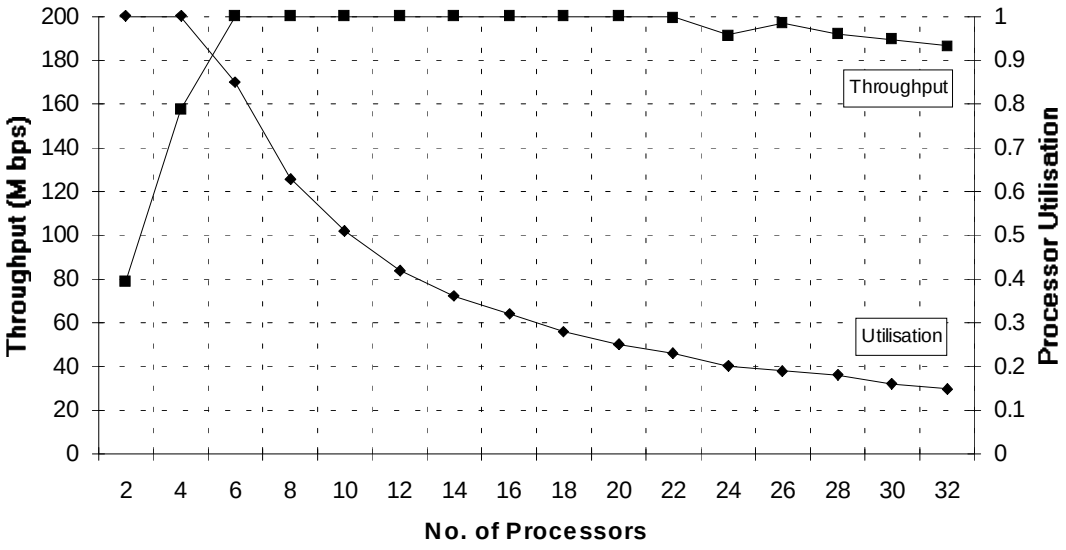

Figure 6 Performance With Processor Speed of 50 MHz

In the next configuration, the processor speed is set to 50 MHz, while varying the network speed. Figure 7 shows that the system performed well with an increasing number of processors as the network speed is increased. As the network speed is increased beyond the 1 Gbps range, it is found that the system begins to saturate. Close examination reveals that this bottleneck is caused by the saturation of the serial access speed of the VRAM, which is able to handle up to a 1 Gbps transfer rate.

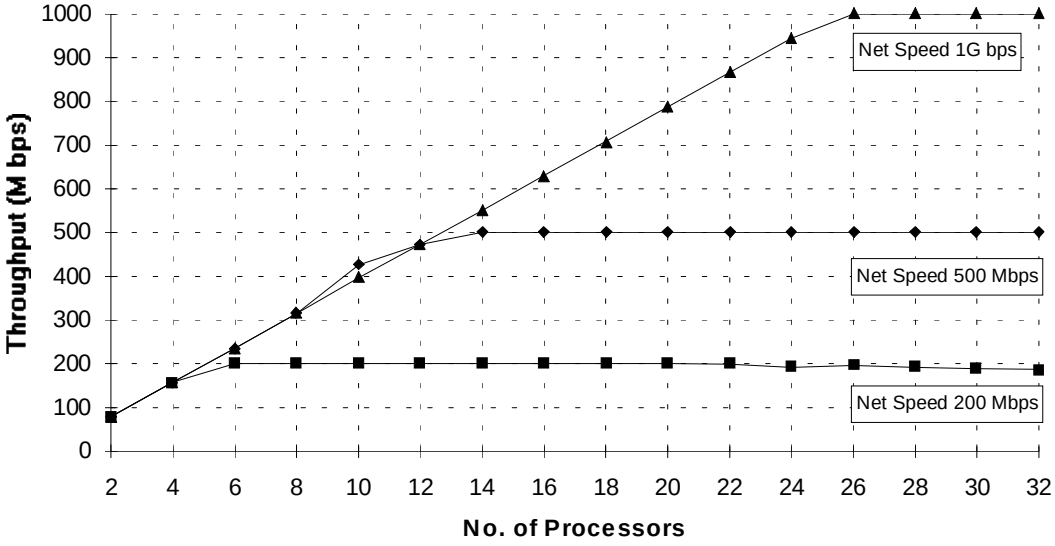

Figure 7 Performance With Varying Network Speed

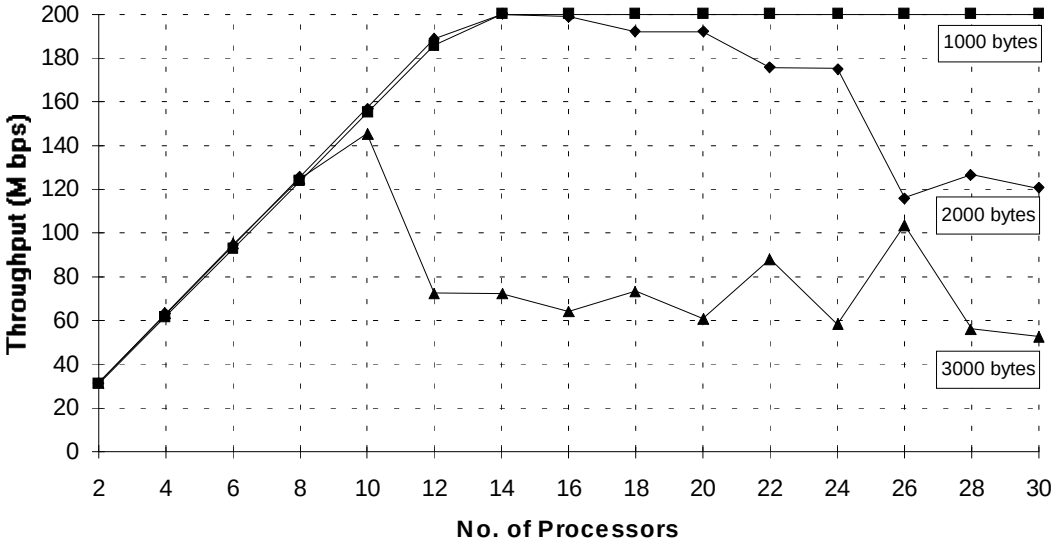

Figure 8 Performance With Varying Packet Size

Figure 8 shows the performance of the system with varying packet sizes, with a processor speed of 20 MHz and network speed of 200 Mbps. The result indicates that the system is able to track the network speed for packet sizes of 1000 and 2000 bytes. However, for packet size of 3000 bytes, the system behaviour becomes unstable as the number of processors increases beyond 10. This is due to the fact that the processing time for each data packet is comparable to the periodic state synchronisation timing. Thus, for each data packet processed, a state synchronisation update is required, resulting in high synchronisation overheads. This result indicates that such unstable behaviour can be

avoided by increasing the periodic state synchronisation timing or by reducing the packet size.

8. CONCLUSION

In this paper, we have examined the use of parallelism for protocol processing to offload the communication bottleneck caused by the development of high-speed networks. Although not specifically designed for protocol processing, a number of features of the transputer and Occam help make protocol implementation efficient both in terms of programming effort and execution speed. Further, the T9000 transputer and C104 router provide a platform with the potential to construct a highly parallel and highly scalable protocol processing system. As a result, an efficient multi-processing architecture based around the T9000/C104 has been presented. The system is designed to exploit a highly parallel computing architecture, in which packet and functional parallelism are applied to protocol processing. Synchronisation between data processors for data ordering is achieved using a technique known as token sequencing. In addition, the architecture uses an out-of-band signalling system based on periodic state synchronisation between end systems.

A simulation of the architecture has been implemented and has demonstrated the advantage of exploiting a parallel architecture for protocol processing. Importantly, the highly scalable features of the architecture offer even greater performance potential as the processing requirements continues to increase.

In future, further analysis on the architecture will be conducted to enable a thorough investigation of the implications of design choices in a truly parallel environment.

ACKNOWLEDGMENT

We would like to thank Rice University, Electrical and Computer Engineering Department, for the provision of NETSIM simulation tool, and in particular, J. R. Jump for his generous assistance.

REFERENCES

1. David D. Clark, Van Jacobson, John Romkey, Howard Salwen, "An Analysis of TCP Processing Overhead," *IEEE Commn. Magazine*, pp. 23-29, June 1989.

2. W. Doeringer, D. Dykeman, M. Kaiserswerth, B. Meiser, H. Rudin and R. Williamson, "A Survey of Light-weight Transport Protocols for High-speed Networks," *IEEE Trans. Commn.*, vol 38, pp. 2057-2071, Nov 1991.

3. A. N. Netravali, W. D. Roome, and K. Sabnani, "Design and Implementation of a High Speed Transport Protocol", *IEEE Trans. on Commn.*, vol. 38, no. 11, pp 2010-24, Nov 1990.

4. G. Chesson, "The Protocol Engine Project," *Unix Review*, Sept 1990.

5. D. C. Feldmeier, "A Framework of Architecture Concepts for High-Speed Communication Systems", *IEEE Journal on Selected Areas in Communications*, vol 11, no. 4, May 1993.

6. David D. Clark and David L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", *Proc. ACM SIGCOMM '90*, pp. 200-8, Philadelphia, PA, Sep 1990.

7. D. C. Feldmeier, "A Survey of High Performance Protocol Implementation Techniques", in *High Performance Networks - Technology and Protocols*, Chapter 2, Ahmed Tantawy editor, Kluwer Academic Publishers, 1993.

8. T. S. Chan and I. Gorton, "A Transputer-based Implementation of HTPNET: A Transport Protocol for Broadband Networks", in *Transputer Applications and Systems '93 Vol 2, Proc. of the 1993 World Transputer Congress*, Aachen, Germany, pp. 889-910, IOS Press, 1993.

9. J. R. Jump, "NETSIM Reference Manual", Rice Univeristy, May 1993.

10. M. D. May, R. M. Shepherd and P. W. Thompson, "The T9000 Communications Architecture", in *Networks, Routers and Transputer*, M. D. May, P. W. Thompson and P. H. Welch (eds.), IOS Press, 1993.

11. M. Simpson and P. W. Thompson, "DS-Links and C104 Routers", in *Networks, Routers and Transputer*, M. D. May, P. W. Thompson and P. H. Welch (eds.), IOS Press, 1993.

12. R. G. Covington, S. Dwarkadas, J. R. Jump, J. B. Sinclair and S. Madala, "The Efficient Simulation of Parallel Systems", *International Journal in Computer Simulation*, vol 1, no. 1, pp. 31-58, 1991.

13. "Transputer Reference Manual," Prentice Hall, INMOS Limited, 1988.

14. SGS-THOMSON, "The T9000 Transputer Products Overview Manual", first edition 1991.