Appeared in Proceedings 6th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), Montreal, QC, Canada, October 1998, pp. 153-160.

Prioritized Multiprocessor Networks: Design and Performance

Govindan Ravindran and Michael Stumm
Department of Electrical and Computer Engineering
University of Toronto
Toronto, Canada M5S 3G4
Email: gravin@eecg.toronto.edu

Abstract

This paper proposes and evaluates prioritized direct shared-memory multiprocessor networks. We use three components to implement prioritized networks, namely, priority-based link arbitration, priority inheritance, and dynamic virtual channels. The two major results from our study are: (i) adding priorities to direct shared-memory multiprocessor networks can lead to reduced average transaction latencies and increased system throughput when running traditional parallel applications, and (ii) a prioritized multiprocessor network can be used to reduce the worst-case latencies of time-constrained traffic when it co-exists with best-effort traffic, without penalizing the average performance of best-effort traffic.

1 Introduction

Adding priorities to direct interconnection networks (of shared-memory multiprocessor systems) can lead to a number of advantages. It can reduce average latencies and improve system throughput. It can be used to support multiple classes of traffic, such as multimedia and regular, best-effort traffic. It leads to much lower variances in latency and hence improved system predictability, which is important for (soft) real-time systems.

Adding priorities to direct networks is surprisingly simple, and involves three main components: (*i*) priority-based link arbitration, (*ii*) priority inheritance, and (*iii*) dynamic virtual channels. With *priority-based link arbitration*, if two or more packets compete for the same idle link, the link will be assigned to the higher priority packet (as opposed to assigning in a round-robin or in a FIFO manner). There are many ways to assign priority to packets. For instance, we can assign priority to packets based on its age, transaction type, or size. Priority-based link arbitration can, however, result in priority inversion, where a lower priority packet may block a higher priority packet that may come behind it in a queue. With *priority inheritance*, a blocking lower priority packet at the head of a queue temporarily inherits the priority of the higher-priority packet behind it [6]. This allows the lower priority packet to obtain the desired link sooner, thereby reducing the queuing delays for higher priority packets. With *dynamic virtual channels*, we dynamically allocate new virtual channel buffers for high priority packets that would otherwise unnecessarily block [7].

In this paper, we show how a connectionless wormhole switched two-dimensional mesh-connected shared-memory multiprocessor network can be extended to support priorities of network packets, and we analyze its performance. Through extensive flit-level simulations, we show how such prioritized networks can significantly reduce latency, improve system throughput and predictability.

In a related earlier work [8], Rexford et. al. propose virtual networks for routing different classes of traffic. Our approach is different in that we use demand driven dynamic virtual channels as opposed to static virtual channels and employ priority inheritance.

2 The Problem

In this section, we illustrate one of the uses of prioritized network, namely to support two classes of traffic. Multiprocessor systems are increasingly being used for multimedia applications, while still serving as data and computation engines. In the backplane networks of such systems, a variety of traffic types will co-exist, ranging from the traffic of sequential and parallel computations (*best-effort traffic*), to the traffic of multimedia audio and video communications (*time-constrained traffic*). These two types of traffic have quite different traffic characteristics and performance requirements. Time-constrained traffic often require a bound on worst-case latency, while a good average-case behavior will suffice for the best-effort traffic arising from regular computations.

Bounds on worst-case latency could be provided

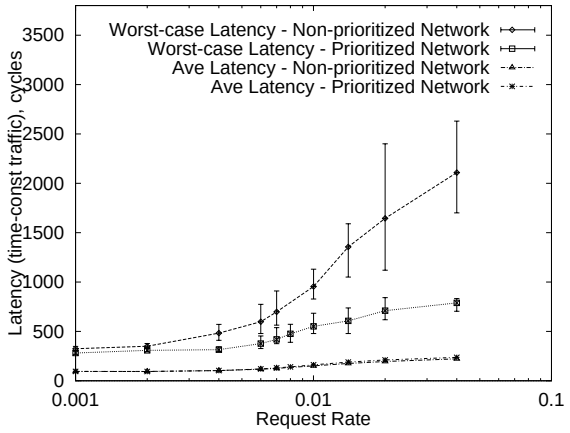

Figure 1: Worst-case and average communication latencies of time-constrained traffic in a 2D 8×8 mesh-connected multiprocessor network. Worst-case latency is shown both for round-robin link arbitration and with dynamic virtual channels. The errorbars show the variances on these values.

if the network is connection oriented and resources can be reserved in advance during a connection set-up phase. A connectionless network, though may not be able to guarantee bounds on worst-case latency, allows for better utilization of network resources among several classes of traffic. Wormhole routed connectionless networks with round-robin link arbitration are used in many of todays multiprocessor routers, and they deliver good average performance. However, worst-case communication latency can be very high and unpredictable as the network load increases. Figure 1 illustrates this. Assuming a workload described in a later section (containing time-constrained and best-effort traffic), the bottom curve plots the *average* communication latency of time-constrained requests as a function of load rate of best-effort requests for a 2D 8×8 mesh network. The top curve plots the *worst-case* latency of the time-constrained traffic for the same workload. We used the batch-mean analysis method [4], where the average latency is computed as the grand average of all batch averages and the worst-case latency is computed as the average of all batch worst-case latencies. For the worst-case latency, the top end of the errorbars represents the global worst-case (over all batches), while the bottom end represents the global best (over all batches) of the worst-cases. It is apparent that the worst-case latencies and their variance increase significantly as the load increases.

The curve in the middle plots the worst-case latencies of time-constrained traffic for the same workload, but for a network that uses the techniques proposed in this paper. It is clear from this curve that the techniques are effective in reducing worst-case latency and its variance without the need for bandwidth reservation. While our goal is to reduce the worst-case latency of time-constrained packets, we wish to do so without unnecessarily penalizing best-effort traffic. By routing time-constrained traffic mainly through dynamically assigned channels, we reserve a set of primary virtual channels, which we refer to as virtual channel 0 or VC-0, for best-effort traffic. This prevents performance deterioration of best-effort traffic even when there is a moderately high level of time-constrained traffic.

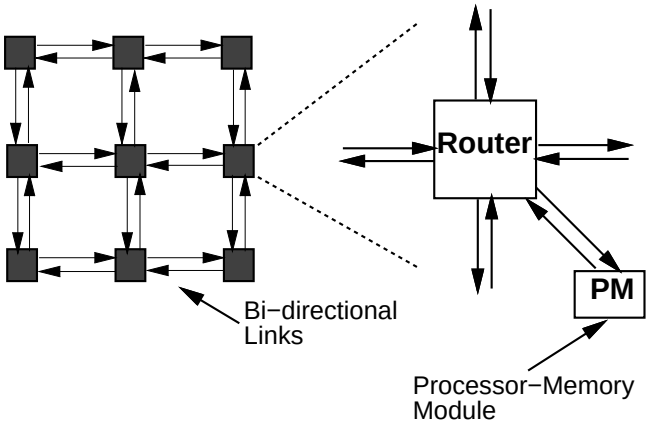

Figure 2: A 2D mesh system with 9 processors.

3 Simulated System

For our study, we assume a 2-dimensional, mesh-connected, shared-memory multiprocessor. Figure 2 shows the network for a system with 9 processors. Each processing module (PM) contains a processor, a local cache and a portion of the main memory. The connection between each pair of adjacent nodes is bidirectional, implemented as two 32-bit wide unidirectional channels and no end-around connections. This topology allows minimal deadlock free x-y routing that does not require virtual channels [1]. This allows us to use virtual channels to route higher priority traffic to improve system throughput [2]. We assume wormhole switching, where a packet is sent as a contiguous sequence of flits with the header flit containing the routing and sequencing information [3].

The system provides a flat, global (physical) address space, and each PM is assigned a unique contiguous portion of that address space, determined by its location. All processors can transparently access

all memory locations in the system. The target memory is determined by the address of the memory being accessed. Local memory accesses do not involve the network, while remote memory accesses require a request packet to be sent to the target memory followed by a response packet from the target memory to the requesting processor. For time-constrained traffic, the PM acts both as a storage node responsible for storing multimedia data and as well a network node that initiates time-constrained requests [5]. The packets are of variable size[1] and are transferred in flits, bit-parallel, along a unique path in the network.

In a mesh-connected system, we refer the router that connects a PM to the mesh as *Network Interface Controller* (NIC). A NIC (with virtual channel buffers) for a bidirectional mesh is schematically shown in Figure 3a. It is modeled as a 5×5 crossbar switch with four input/output links from and to its four direct neighbors and one input/output link from and to the local PM. The input links have FIFO buffers to store flits that are blocked in the network.

The NIC performs basic switching, routing and flow control functions. It examines the header flit of a packet to determine which output link the packet should be forwarded to. The NIC also does proper arbitration if there are competing requests for an output link. The arbitration policy could be round-robin, priority-based or both. In our study, we assume priority-based arbitration with two levels of priority: a high and a low priority. If a requested output link is not available, then the requesting flit is blocked and stored in the corresponding input buffer. It is assumed that the NIC can connect all inputs to outputs in a single network clock cycle. Once a switch connection between an input and output link is established, it is broken only after the last flit of a packet has been transferred. We assume *buffered* wormhole switching with NIC buffer size large enough to store 3 flits [7].

4 Static Virtual Channels

A network with virtual channels organizes the flit buffers associated with each physical channel into several virtual channels. Virtual channels increase physical channel utilization, and thus network throughput, because any blocked packet that spans several nodes occupies only one virtual channel, and can be bypassed using any of the other virtual channels associated with a physical channel. The virtual channels associated with a physical channel arbitrate for physical channel bandwidth on a flit-by-flit basis. With *static virtual channels*, the number of virtual channels per physical channel remains constant. Figure 3a shows a mesh NIC with two virtual channels per physical channel, which remains constant.

At the receiving side of a node, the routing algorithm first assigns an incoming packet to an output physical channel and then to a virtual channel. If virtual channels are being used for deadlock free routing, then the choice of virtual channel is dictated by the routing protocol; otherwise, another allocation scheme is used or any free virtual channel associated with the physical channel is chosen. Once a packet is assigned a virtual channel, flit-level flow control is used to advance the packet across the switch and the physical channel.

Hardware support for static virtual channel flow control requires status registers at the transmitting and the receiving side of a node [2]. The transmitting node contains a status register for each virtual channel on the corresponding receiving node. The status register normally includes a bit to indicate whether the virtual channel is active or idle and a count of the number of free virtual channel buffers. The active/idle bit is used to prevent interleaving of the flits of different packets. The receiving node contains a status register for each virtual channel that contains information such as the state of the channel and optionally, input and output virtual channel pointers. The status register storage requirement per physical channel is:

$$S_{pc} = N(\log(B_{vc}) + 1) + N \qquad (1)$$

where the first and second term represent the storage requirement at the transmitting and receiving side of a node, respectively. B_{vc} is the number of flit buffers per virtual channel, and N is the number of virtual channels per physical channel. For $N = 4$, and $B_{vc} =$ 4 flits, the status buffer storage requirement is 16 bits.

Adding virtual channels requires a few additional wires in the physical channel to identify the virtual channel for each transmitted packet in the forward direction and to indicate the availability of buffers to the transmitting node in the reverse direction. The virtual channel buffer counter at the transmitting side is incremented each time a flit is transmitted to the neighboring node and decremented when the neighboring node signals that it has forwarded a flit and thus freed up buffer storage by back propagating a *freed* bit along with the virtual channel identification. The extra channel width overhead for supporting virtual channels in a network with 32-bit phits with 4 static virtual channels per physical channel is: 2 bits

[1] Six main packet types are simulated, namely read request, read response (cache-line size), write request (cache-line size), write response, time-constrained request, and time-constrained response (cache-line size).

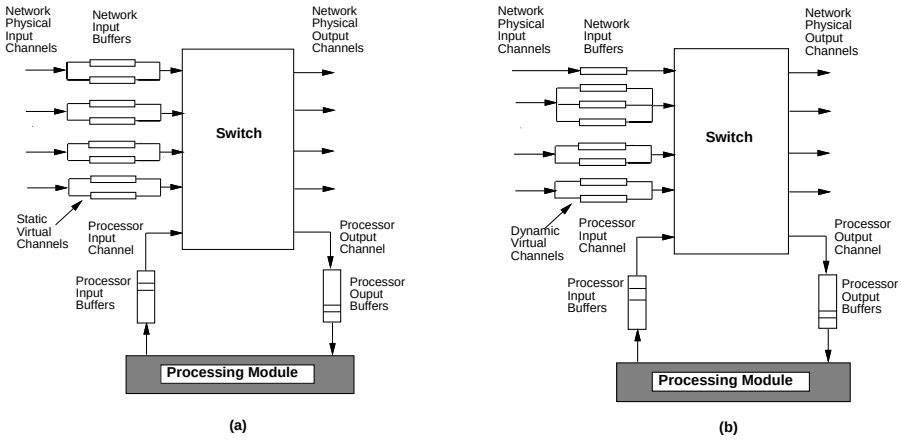

Figure 3: Mesh Network Interface Controller with (a) static and (b) dynamic virtual channels.

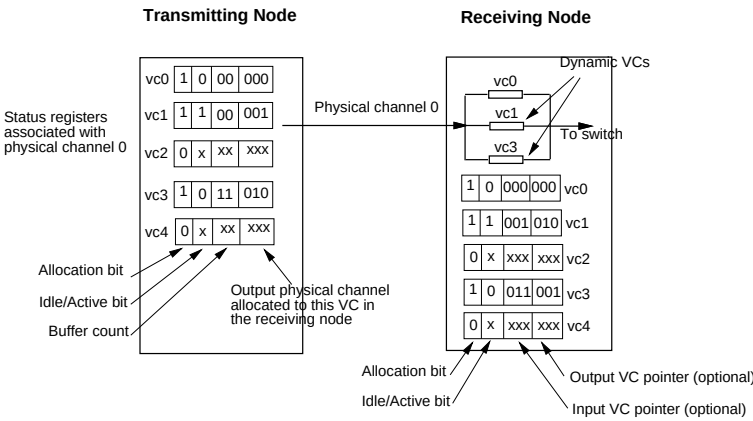

Figure 4: Hardware support for dynamic virtual channel flow control is illustrated for one physical channel between a transmitting and a receiving node.

to transmit the virtual channel id in the forward path, 2 bits to transmit the virtual channel channel id in the reverse path, and a *freed* line.

5 Dynamic Virtual Channels

Dynamic virtual channels are similar to static virtual channels in that they are multiplexed over a single physical channel and each of these dynamic channels have independent FIFO buffers of the same size. However, unlike static virtual channels, virtual channels in this case are allocated dynamically from a common pool. Thus, the number of virtual channels per physical channel varies over time, with the minimum number per physical channel being 1. Figure 3b shows a NIC with dynamic virtual channels.

In our case, a new virtual channel is allocated dynamically, if possible, for a high priority packet that would otherwise unnecessarily block. The number of dynamic channels allocated per physical channel thus varies depending on the contention for the physical channel. Routers using dynamic virtual channel allocation prevent head-of-line blocking effectively, where a packet waiting for a blocked link is itself blocking another packet behind it whose target output link is free.

We assume that the total number of virtual channels that can be allocated in a NIC is constant. Initially there is one virtual channel per physical channel, which we refer to as VC-0. A virtual channel is allocated for a packet by the flow-control logic at the transmitting side of a link, which transmits the dynamic virtual channel number along with the packet (similar to the static virtual channel allocation case). At the receiving side of a node, when a packet arrives, it is buffered in the specified virtual channel buffer (if the virtual channel has been already allocated to the physical channel). When the specified virtual channel does not exist, it will be allocated from a common buffer pool. A dynamic channel, once allocated, is released only when it contains no more data. In the rare case when there are no free common pool buffers[2] for an incoming packet, then it cannot be assigned the specified virtual channel and the packet (header flit) is dropped and a *drop* signal is asserted. The transmitting node then retransmits the header flit when the drop signal is deasserted. This requires the transmitting node to keep a copy of the header flit when a dynamic virtual channel is requested so that it can

[2]This can happen when two or more arriving packets at different physical channels require new virtual channels at the same time and only some of the requests could be granted.

later be retransmitted if necessary. This has no performance impact on the system, as it is equivalent to blocking a flit for an extra cycle.

Dynamic virtual channels can be implemented with a simple extension to the hardware used to support static virtual channels. Figure 4 presents the hardware required for implementing dynamic virtual channel flow control for one physical channel between a transmitting and a receiving node. Similar to the static virtual channel case, the transmitting node contains a status register for each virtual channel on the receiving side. The number of such status registers is equal to the maximum number of possible dynamic virtual channels. The status register contains an allocation bit to identify whether the virtual channel has been allocated to a physical channel and, if allocated, a bit to indicate whether it is idle or active, and a count of the number of free virtual channel buffers. In addition to the above, to avoid head-of-line blocking, 3 bits are required to store the output physical channel number assigned to the packet at the head of virtual channel buffer in the receiving node.

The receiving side contains a status register for each virtual channel; the register contains an allocation bit and/or an idle/active bit. The status register storage requirement per physical channel is therefore:

$$S_{pc} = N_{max}(\log(B_{vc}) + 2 + 3) + 2 \cdot N_{max} \qquad (2)$$

where the first and second term represents the storage requirement at the transmitting and receiving side of a node, respectively, B_{vc} is the number of flit buffers per dynamic virtual channel, and N_{max} is the number of maximum virtual channels that can be assigned. For $N_{max} = 4$, and $B_{vc} = 4$ flits, the status buffer storage requirement becomes 36 bits.

With respect to channel width overhead, similar to the static virtual channel case, we need to identify the virtual channel number both in the forward direction that is transmitted along with the packet and in the reverse direction that is transmitted along with the *freed* signal. In addition, an extra wire is required for the *drop* signal that is asserted when a header flit is dropped.

6 Simulator

The simulator we use reflects the behavior of the system we simulate at the register-transfer level on a cycle-by-cycle basis. It was implemented using the *smpl* simulation library [4]. The batch means method of output analysis was used, with the first batch discarded to account for initialization bias. A base version of the simulator was validated against measurements taken from the Hector prototype, a hierarchical slotted ring architecture [9]. The base simulator was then extended to model meshes and switching techniques such as wormhole switching.[3]

Our measures of performance are system throughput (in requests completed per processor cycle), and worst-case and average round-trip memory access latency (in processor clock cycles). We assume that the network clock cycle is twice the processor clock cycle. The average round-trip latency is computed as the grand average of all batch averages, while the worst-case round-trip latency is computed as the average of maximum round-trip latency of all batches.

A processor is allowed to have four outstanding requests, before it is required to block for a reply. This parameter is used to model processors with prefetching and/or multi-threading. For best-effort traffic, we assume the probability of a request being a read is 0.7 (the remaining being write requests). The batch termination condition is that all processors have to complete a minimum number of requests.

7 Prioritized Direct Networks

In our implementation of a prioritized direct network, one virtual channel, VC-0, is initially statically assigned to each physical channel. In addition, virtual channels are allocated dynamically (from a pool) to a physical channel. Low-priority packets may only use VC-0s, while high priority packets use dynamically assigned channel(s).

We use a three step process to allocate output links. Output links are first allocated to high priority packets buffered in dynamic virtual channels. Among competing high priority packets, we allocate the output link to the oldest one. Second, we assign output links to high priority packets, if any, at the head of the processor input queue. Finally, lower priority packets in the VC-0s and at the processor input queues are assigned output links in that order. Since we have independent virtual channels for time-constrained traffic in the network, we need to apply priority inheritance only at processor input queues, as that is the only place where priority inversion can occur.

In this section, we show how effective priority networks are in reducing latency and in improving system throughput and predictability. We do this by simulating a 2-dimensional mesh-connected network, extended with priority-based link arbitration, priority inheritance, and dynamic virtual channels. Although our evaluations are for two priority levels, a high and

[3] For the mesh simulator, the processor and memory modules are essentially the same as in the ring simulator with new NIC modules that incorporate switching, routing and flow-control.

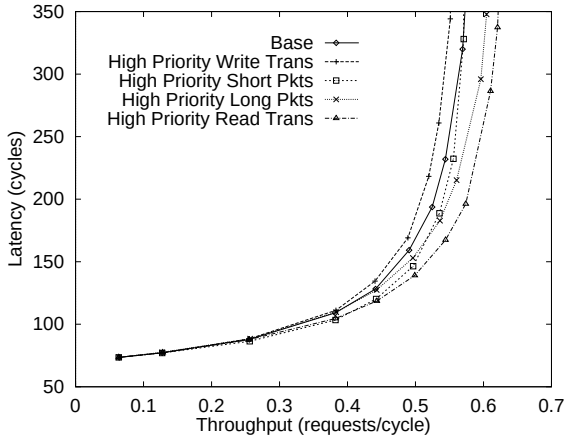

Figure 5: Throughput versus latency for a 64 processor 8×8 wormhole switched prioritized network. The base case is for a non-prioritized network with no dynamic channels. For the other curves higher priority is given to packets based on its size (longer or shorter) or its transaction type (read or write).

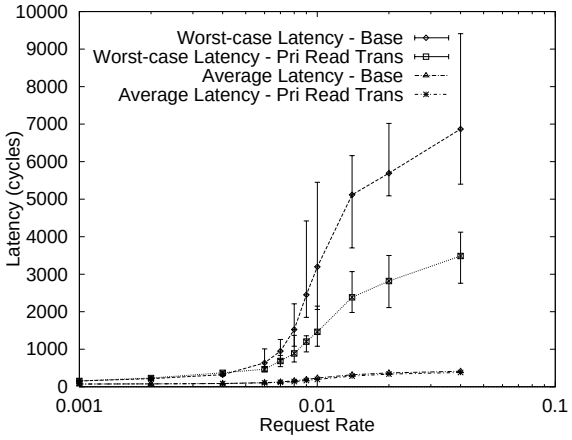

Figure 6: Worst-case latency versus request rate for a 64 processor 8×8 wormhole switched prioritized network. Curves are drawn for a base case of a non-prioritized network with no dynamic channels and for a network with dynamic channels where read transaction is given higher priority.

a low priority level, it could be extended to multiple priority levels. We also show that dynamic virtual channels can be used to support multiple classes of traffic. For this purpose, we consider two traffic classes, namely best-effort traffic and time-constrained traffic. We show that the priority network is effective in reducing the worst-case communication latency of time-constrained traffic, while not penalizing best-effort traffic.

7.1 Priority Traffic for Traditional Applications

Even with no time-constrained traffic, it can make sense to assign priorities to different classes of packets if it benefits that class or the traffic overall. For example, in a shared-memory multiprocessor, one can consider giving a higher priority to large packets containing data or to shorter packets containing requests or acknowledgments. Large packets consume more network resources (e.g., links and buffers) than short packets, and when a large packet is blocked in the network, it will unnecessarily block other packets, thereby reducing system throughput. By giving priority to large packets, they will be removed from the network sooner, thereby reducing the number of packets they can block. On the other hand, by giving priority to short packets, we prevent them from being unnecessarily blocked by large packets.

It is also possible to prioritize packets according to transaction type i.e., read and write transactions. Figure 5 presents the throughput-latency curves for five different cases: a base case of a non-prioritized network with no dynamic or static channels, and a prioritized network with high priority given to large packets, short packets, read transactions, and write transactions. This is for a 64 processor 8×8, mesh-connected system with 32 byte cache lines and a workload with a uniformly distributed memory access pattern. For the prioritized network, we assume a virtual channel buffer size of 3 flits and that the maximum number of dynamic virtual channels is 4. For the base case, the network input buffer size of 6 flits is twice as large as that of the prioritized network, under the assumption of equal memory resources.

As can be seen from Figure 5, the highest throughput is achieved when read transactions (i.e., read request and read response packets) are given high priority. Also, giving priority to large packets results in better performance than giving priority to short packets, but giving priority to write transactions results in a poor performance. A possible explanation for this is that since the number of read transactions is far higher than the number of write transactions, giving priority to read transactions will result in a higher dynamic channel buffer utilization when compared to high-priority write transactions.

To measure the impact of prioritized networks on the predictability of the system, we plot worst-case latencies in Figure 6 for both the non-prioritized network and for the prioritized network with high priority read transactions. For comparison purposes, we also

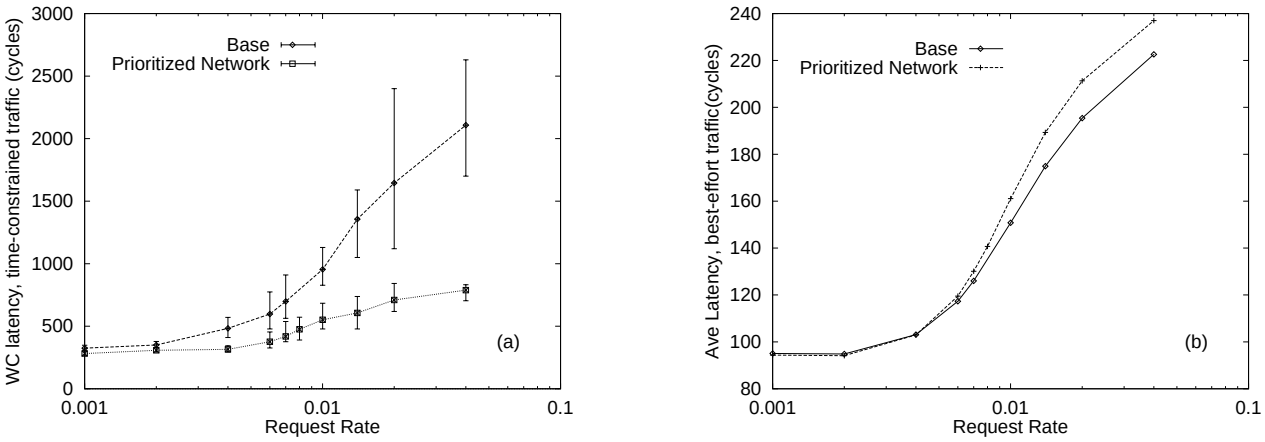

Figure 7: (a) Worst-case latency of time-constrained requests and (b) average latency of best-effort requests, both as a function of best-effort request rate for an 8×8 64 processor wormhole switched prioritized network. Curves are for a base case of a non-prioritized network with no dynamic channels and for a network with dynamic channels, with priority given to time-constrained packets.

present the average latency curves. It is clear that though the average latencies are small, the worst-case latency for the non-prioritized network can be as high as a factor 50 higher than for the priority network. The unpredictability of the worst-case latency values is shown by the length of the errorbars. The prioritized network substantially reduces the average worst-case latency, and it reduces the variance by more than a factor of 2, thereby improving the predictability of the system.

7.2 Time-constrained Traffic

In this section, we consider a mix of two classes of traffic: (*i*) *best-effort* traffic with uniformly distributed destinations and an exponentially distributed inter-arrival time between requests, and (*ii*) *time-constrained* traffic with destinations uniformly distributed, but with a fixed inter-arrival time between requests, as seen in multiprocessor video servers [5]. In our simulations, a processor is allowed to have 2 outstanding best-effort requests and 2 outstanding time-constrained requests for a total of 4 outstanding requests, before it is required to block for a reply.[4] For best-effort traffic, we assume that 32 byte cache lines are being transferred. The batch termination criterion is that all processors have to complete both a minimum number of best-effort requests and a minimum number of time-constrained requests. In all our experiments we vary the request rate of best-effort traffic, and we measure the worst-case latency of time-constrained requests and the average latency of best-effort requests. The inter-arrival time of time-constrained requests is fixed at 1 in 1000 processor cycles.

Figure 7a presents the average worst-case communication latency (over all batches) of time-constrained requests. Errorbars indicate the absolute maximum and minimum values over all batches. There are two curves: the top curve represents a non-prioritized network with no dynamic channels, whereas the bottom curve is for a prioritized network with four dynamic channels per node, giving priority to time-constrained traffic. It is clear that a prioritized network is effective in reducing the worst latency of time-constrained requests more than 50%. In particular, the prioritized network is effective in reducing the absolute maximum worst-case latency of time-constrained requests, thereby improving the predictability of the network.

Figure 7b presents the average latency of the best-effort requests as a function of best-effort request rate. The graph shows that giving priority to time-constrained requests does not significantly worsen the average latency of best-effort requests.

We now consider a non-uniform *bit-complement* best-effort traffic pattern. The bit-complement traffic congests the center of a 2D mesh network and significantly affects the worst-case latency as shown in Figure 8a. A prioritized network can again be effective in reducing by an order of magnitude the worst-case latency of time-constrained requests. Another benefit of

[4] In this model, the time-constrained and best-effort requests are interleaved and can be assumed to be equivalent to having a main processor and a co-processor with the former issuing best-effort requests while the latter issuing time-constrained requests independent of each other.

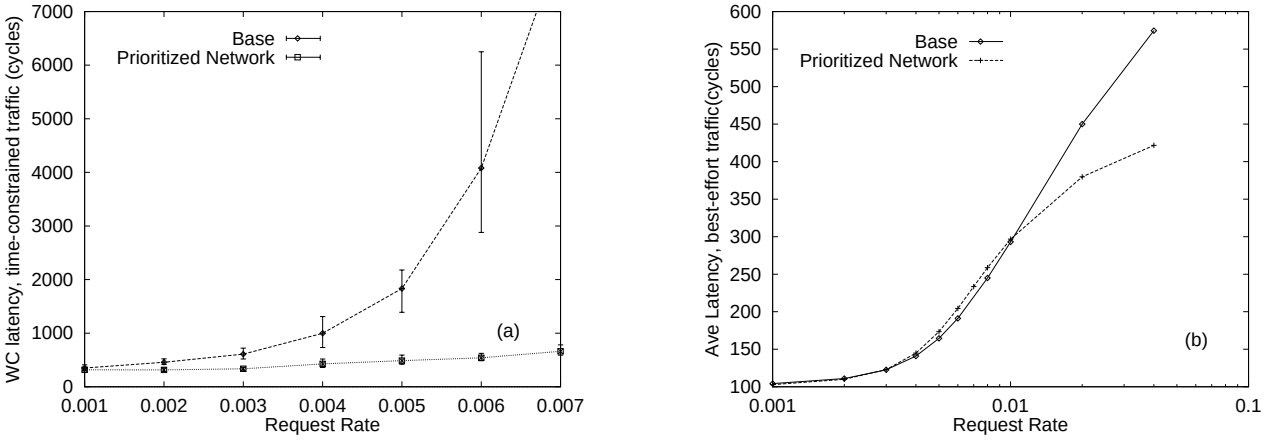

Figure 8: (a) Worst-case latency of time-constrained requests and (b) average latency of best-effort requests, both as a function of best-effort request rate for an 8×8 64 processor wormhole switched prioritized network. A non-uniform bit complement memory access pattern is used for best-effort requests. Curves are drawn for a base case of a non-prioritized network with no dynamic channels and for a network with dynamic channels, where time-constrained packets are given priority.

the prioritized network with a non-uniform best-effort memory access pattern, is a significant reduction in the average latency of best-effort requests at high request rates when compared to the non-prioritized network (see Figure 8b).

8 Conclusion

In this study we proposed and evaluated prioritized connectionless shared-memory multiprocessor networks. In our implementation of prioritized networks, we used three main components; priority-based link arbitration, priority inheritance, and dynamic virtual channels. It was shown that a prioritized network can significantly reduce average transaction latencies and improve system throughput when running traditional parallel applications. It was also shown how a prioritized network could be used to reduce the worst-case latencies of time-constrained traffic when it coexists with best-effort traffic. One of the key aspects of the prioritized network is that it do not increase the average latency of best-effort traffic while improving the average latency of time-constrained traffic, independent of the best-effort traffic pattern.

References

[1] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Computers*, vol. C-36, No. 5, pp. 547-553, May 1987.

[2] W. J. Dally, "Virtual-channel flow control," *IEEE Trans. on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, March 1992.

[3] W.J. Dally and C. L. Seitz, "The Torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp 187-196, March 1986.

[4] M. H. MacDougall, *Simulating Computer Systems: Techniques and Tools*, MIT Press, 1987.

[5] A. L. Narasimha Reddy, "Scheduling and data distribution in a multiprocessor video server," *Proc. Intl. Conf. on Multimedia Computing and Systems*, pp. 256-263, May 1995.

[6] R. Rajkumar, "Synchronization in real-time systems: A priority inheritance approach," *Kluwer Academic Publishers*, ISBN 0-7923-9211-6 pp. 15-58, 1991.

[7] G. Ravindran, *Performance Issues in the Design of Hierarchical-ring and Direct Networks for Shared-memory Multiprocessors*, Ph.D. Dissertation, Department of Electrical and Computer Engineering, University of Toronto, January 1998. http://www.eecg.toronto.edu/gravin

[8] J. Rexford, J. Dolter and K. Shin, "Hardware support for controlled interaction of guaranteed and best-effort communication," *Proc. Second Workshop on Parallel and Distributed Real-time systems*, pp. 188-193, April 1994.

[9] Z. G. Vranesic, M. Stumm, D. Lewis, and R. White, "Hector: A hierarchically structured shared-memory multiprocessor," *IEEE Computer*, vol. 24, no. 1, pp. 72-78, January 1991.