The Impact of Link Arbitration on Switch Performance *

Marius Pirvu, Laxmi Bhuyan and Nan Ni
Department of Computer Science
Texas A&M University
College Station, TX 77845
E-mail: {pirvum, bhuyan, ninan}@cs.tamu.edu

Abstract

Switch design for interconnection networks plays an important role in the overall performance of multiprocessors and computer networks. In this paper we study the impact of one parameter in the switch design space, link arbitration. We demonstrate that link arbitration can be a determining factor in the performance of current networks. Moreover, we expect increased research focus on arbitration techniques to become a trend in the future, as switch architectures evolve towards increasing the number of virtual channels and input ports.

In the context of a state-of-the-art switch design we use both synthetic workload and execution driven simulations to compare several arbitration policies. Furthermore, we devise a new arbitration method, Look-Ahead arbitration. Under heavy traffic conditions the Look-Ahead policy provides important improvements over traditional arbitration schemes without a significant increase in hardware complexity. Also, we propose a priority based policy that is capable of reducing the execution time of parallel applications. Lastly, we enhance the arbitration policies by a supplemental mechanism, virtual channel reservation, intended to alleviate the hot-spot problem.

1. Introduction

The performance of the interconnection network is critical in multiprocessor systems because the network latency contributes significantly to the overall execution time of an application. The factors that affect network performance are the switching technique, buffer management and physical link arbitration. Most of the research that deals with switch design concentrates either on the switching technique or on the buffer management. The third factor, arbitration, is either briefly mentioned or completely overlooked. In this paper we demonstrate that arbitration can play an important role in the performance of a system. Moreover, this role is expected to increase as current switches become more and more complex, have more virtual channels and more input ports.

In our study we compare several arbitration policies and show how they affect the network latency and throughput. We design a novel arbitration method, Look-Ahead, which outperforms traditional policies such as First Come First Served or Round Robin, by keeping a better balance among the link loads. We also show how to apply a simple priority based scheme to reduce the execution time of applications in shared memory multiprocessors. Furthermore, we refine our arbitration policies by a supplemental mechanism, virtual channel reservation, intended to alleviate the hot-spot problem. This mechanism works well in conjunction with any arbitration policy and gives performance benefits even in the absence of an explicit hot-spot.

Related arbitration studies most often look into bus arbitration issues [7, 14, 10]. There are only a handful of papers that discuss link arbitration for switches in interconnection networks. However, in these papers the analysis method is typically modeling [1]. To the best of our knowledge this is the first paper to study switch arbitration in more detail using simulations. Our main results are based on simulations with a synthetically generated workload. We further validate our results through execution driven simulation of a few parallel programs. Our simulated architecture reflects the current state-of-the-art interconnect and processors. The switch we use in our simulations is based on wormhole routing with virtual channels [4] and employs a DAFC (dynamically allocated fully connected) [5] design which is known to perform the best compared to other alternatives. We chose this configuration so that the only source of improvement remains a better arbitration scheme. Current state-of-the-art switches use simple, traditional arbitration policies (e.g. first-come-first-served and round robin). We prove that the implementation of alternative arbitration policies, such as shortest message first or our Look-Ahead and Priority schemes, is definitely worthwhile. Each of these arbitration techniques is focused on optimizing a particular performance metric and addresses a different network problem.

Shortest message first, a simple but often ignored arbitration policy, is designed to improve the average message latency. It performs well when the traffic is composed of messages of different sizes. By promoting shorter packets, we achieve a secondary beneficial effect: the virtual channel availability is increased due to the faster rate at which virtual channels are freed.

Our Priority arbitration policy is targeted towards reducing the latency of critical messages. Even if the average message latency may increase, the overall execution time is reduced. This policy builds on priority-based mechanisms commonly used in real-time systems. We use this technique in shared memory multiprocessors giving priority to read

*This research has been supported by NSF Grants CCR 9622740 and CCR 9810205

memory accesses over write accesses.

The Look-Ahead policy we propose attempts to increase the link throughput. It achieves a better load balancing among the physical links by using information provided by the neighbor switches. The information we exchange is the value of the link load. To increase the parallelism of flits, the arbiter will prefer a packet going to a lightly loaded link. This policy can be applied to any topology and is scalable because it involves communication only between neighbor switches.

All of the above policies can be used either as stand alone or as refinements to existing policies. We show that they can be successfully combined into more powerful policies covering a wide range of network traffic conditions.

However, none of them is capable of preventing the lane hogging problem [8] common in hot-spot situations. We introduce an additional mechanism, virtual channel reservation to address this problem. Lane hogging can occur in traditional implementations of wormhole routing with virtual channels because any worm can occupy any virtual channel irrespective of the destination of the packet. This can lead to the case where all the channels are used up by worms going to the same destination, thus blocking the way of other worms. Our reservation mechanism prohibits the arbiter from choosing a worm which could hog the next input port. This method resembles the flow group implementation in ATLAS I chip [9] where identically destined flits (thus belonging to the same flow group) are not allowed to reside at the same time in an input port. In contrast to this method, we restrict the number of worms (not flits) that go to the same destination. More importantly, we demonstrate that this restriction should not be fixed across the whole network, but should vary according to the position of the input port in the interconnect.

To summarize, the paper makes the following original contributions:

- Quantifies the importance of arbitration in the performance of a switch and investigates the conditions under which arbitration has a greater impact on performance
- Evaluates the behavior of several arbitration policies
- Proposes a new arbitration scheme, Look-Ahead, that aims at evenly distributing the load among the physical links
- Shows how to implement a Priority arbitration scheme in the context of shared memory multiprocessors
- Introduces an arbitration enhancement, virtual channel reservation, designed to avoid the performance bottleneck due to lane hogging

The rest of the paper is organized as follows: Section 2 describes our test-bed and other architectural assumptions. Section 3 presents various arbitration policies currently existent and proposes a new policy called Look-Ahead. The arbitration policies are compared in terms of performance in Section 4. Section 5 introduces the idea of virtual channel reservation as a method of preventing the negative effects of the hot-spot. Finally, Section 6 concludes the paper.

2. Simulation Environment

Our simulator is based on RSIM [12], but we completely changed the network part to allow for a cycle by cycle simulation of various arbitration policies. The architecture we simulate is a shared memory machine with 16 nodes connected by a 4×4 mesh like in Figure 1(a). Each node is

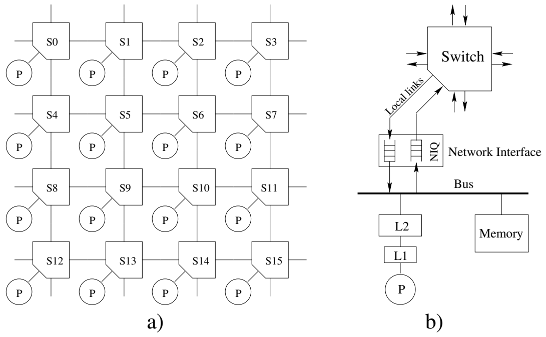

Figure 1. Simulated architecture

composed of a processor, two levels of write-back cache, a memory module, a network interface and a switch (see Figure 1 (b)). We simulate a state-of-the-art 4-way superscalar processor with branch prediction, out-of-order speculative execution, non-blocking reads and release consistency shared memory model. The only simplifying assumption we make is that instructions are considered to always hit in the I-cache of the processor. The instruction window of the processor has 64 entries and the functional units have latencies similar to the UltraSPARC processor. The memory is four way interleaved and has an access time of 18 cycles. The bus connecting the various components is a split transaction bus with a width of 16 bytes; its clock is three times slower than that of the processor. All the other characteristics of our simulated architecture are summarized in Table 1. The size of the L2 cache is smaller than one can find in today's machines to account for the rather small problem size that we run.

To accurately measure the contention in the network we employ a cycle by cycle simulation. The switch at each node is based on our design proposed in [11]. It has 4 input ports, 4 output ports and a pair of local links (1 input, 1 output) for the communication with the processing unit. It

Switch parameters	
flit size	8 bytes
link width	2 bytes
switch cycle	4 pcycles
wire delay	1 pcycle
routing delay	4 pcycles
routing algorithm	X-Y
virtual channels	4 per input
buffer size per port	32 flits
Cache parameters	
L1, L2 size	16 K, 128 K
L1, L2 associativity	2 way, 4 way
L1 latency	1 pcycle
L2 latency	8 pcycles pipelined
L1, L2 cache line	64 bytes
L1, L2 write policy	write back, write allocate

Table 1. Default architectural parameters

uses wormhole routing and has four virtual channels (VC) at each input port. Such a small number of VCs is common. The Spider [6] and Cavallino [2] routers have both four VCs per input while the switch used in Cray T3E network has five. However, in our case the input buffer is dynamically allocated using a circular buffer [11]. Moreover, to improve performance, one flit per VC is reserved and only the rest of the input space is shared. In some experiments the total size of the input buffer is varied between 16 and 64 flits; in others we use the default value of 32 flits. This is only half of the value used in the Spider chip or Cray T3E routers. We think that the dynamic buffer allocation enabled by our switch design justifies the use of a smaller input buffer. The switch is fully connected which permits us to simultaneously read several flits out of an input port and pass them to the corresponding outputs. A secondary benefit deriving from this feature is that the design of the arbiter is simpler.

The physical links between switches are considered to operate at processor frequency, while the core (the switch) is four times slower. Thus, a *switch cycle* is four times the wire cycle. Every packet is divided in flits of 64 bits each. Because the width of the links is only 16 bits, a flit is further divided in four *phits*. Hence, the transmission of a flit is achieved by four consecutive phit transfers. To assemble flits at reception and disassemble flits for transmission two special purpose registers (RxR and TxR) are kept at each input port. The arbitration is performed at flit level, every switch cycle. The flow control is also maintained at flit level and, similar to Spider [6] and ATLAS I [9] chips, is done using credit flow information [3].

In our experiments we employ two simulation methodologies:

A. Synthetic workload. In this case the processor characteristics are irrelevant. The packets are generated with a given frequency and pushed in the network interface queue (NIQ) whose size is large enough to avoid overflow. From the NIQ the packets are absorbed one by one by the network and routed towards their destinations. For each generated packet its destination is randomly determined. The performance metrics we employ are throughput and message latency.

In parallel machines periods of very high levels of traffic are mixed with periods when very few messages are pushed into the network. Thus, it is interesting to analyze the network behavior during congestion phases. In our experiments packets are generated at a saturation rate so that, on average, for each processor a flit is created each cycle. Due to congestion the NIQ has the tendency to build up. If simulations were continued forever the latency would increase without a limit. Instead, we abruptly stop the simulation after 32000 packets have been received. Due to this technique the absolute value of the latency on the graphs is not very meaningful. However, the relative performance between the various arbitration techniques is an accurate indicator of how good an arbitration method is when compared to the others.

B. Execution driven simulation of shared memory applications. In this case the only metric that we employ is the execution time. As applications we use three kernels: FFT and RADIX sort from the SPLASH-2 benchmark suite [13], and MATMUL (matrix multiplication). FFT is run for 2^{16} complex points, RADIX for 2^{18} integers and MATMUL for 128×128 matrices. The time has been measured only for the parallel part of the applications.

The packets pushed into the network are of two types: requests and replies. The requests are usually short. In our case they have a length of two flits. As for the replies, if they carry a cache line, they have 10 flits in length; otherwise they have also two flits (acknowledgments fall in this category for example).

3. Presentation of the Arbitration Policies

In the following we describe some known arbitration policies used in current state-of-the-art switches and propose how to use a priority policy in the context of shared memory multiprocessors. More importantly, we introduce a new and better policy called Look-Ahead.

3.1. Round Robin

One of the most popular arbitration policy is *Round Robin*. Here, the arbiter scans all the channels and the first one that is ready to transmit, say channel i, is selected to send one flit. The next cycle the scanning process starts with channel $i + 1$. This method is evidently fair and no starvation can occur. However, the network latency tends to be uniformly increased for all the worms. To alleviate this negative effect we can make the following amendment: the scanning process will move to the next ready VC only if the current VC has just sent a tail flit. In other words, the arbiter will prefer the worm that is currently flowing out of the switch. When this worm has drained out completely, the simple Round Robin policy is used. We will refer to this scheme as *Round Robin - Keep Flow*. A slightly modified variation of this scheme is used in the routers for the Cray T3E network.

3.2. First Come First Served

In *FCFS* the oldest worm has priority. A possible implementation is the following: an 'age' counter is associated with each VC. When a new worm enters a VC, the age counter is set to 0. Afterwards, the age is incremented at a programmable rate. The arbiter will prefer the worm with the highest age. Inherently this arbitration policy is starvation free. As in the Round Robin case we can have a *FCFS - Keep Flow* method where the worm that is currently flowing out of the switch is given priority. The Spider chip [6] uses an 8-bit age counter to implement FCFS. However, their implementation differs from ours in that the age is carried along by the worm and is not reset when the worm enters a new switch.

3.3. Shortest Message First

From the process scheduling theory it is known that shortest-job-first achieves a very good response time. The equivalent policy for arbitration is *Shortest Message First* (SMF) which tries to promote the worms with the fewest flits still to be transmitted, thus minimizing the message latency. A second motivation of the SMF policy is increased VC availability. This advantage arises when the input buffer is fairly large and only a single worm can use a VC at a time. For such switches there might be situations where, even though there are sufficient buffers available, a header flit is blocked because the next switch does not have a free VC. Hence, the arbiter should try to free virtual channels as fast as possible and this is exactly what SMF does. However, the method is not fair and, if special measures are not taken, starvation can occur. To the best of our knowledge this arbitration method is not used in the current switches.

3.4. Priority Arbitration for Shared Memory Multiprocessors

When arbitration is based on priority, each packet is assigned a priority level. Packets with higher priorities are always selected first. When multiple packets have the same priority, the FCFS method is used to break the tie. To avoid starvation some special measures must be taken. For instance, in the Spider chip [6] when a packet accumulates a certain age, its priority level is increased.

Traditionally, this arbitration policy has been used in the context of real time systems. We will show how it can be applied with success to shared memory multiprocessors that employ relaxed memory consistency models. In such a machine the latency of write operations can be hidden to a large extent. Read operations are more critical because the processors will soon stall in the absence of the desired data. Therefore, it seems natural to give priority to read operations over write operations. In our implementation we use just two levels of priorities. Read requests and read-modify-write (RMW) operations issued by the processor have priority 1. Write requests have priority 0. All the other consistency messages and replies inherit the priority of the message they originated from. For instance, if due to a write request the memory needs to send some invalidation messages, all these invalidations together with their corresponding acknowledgments will inherit the priority of the write (i.e. 0). If the same invalidations are caused by a RMW request they will have priority 1. Such a simple scheme is far from being perfect. For example, the command given to processors to leave a synchronization barrier is essentially a write operation. Delaying this write request can negatively affect the performance of applications. Such subtleties are not the goal of this paper and will not be discussed here. Rather, we keep the implementation simple to show the potential of such an arbitration scheme in the context of shared memory multiprocessors. Its evaluation will be done using execution driven simulation of a few parallel applications. We will not use this arbitration policy with synthetic workload because it is impossible to compare it to other policies: of course, the high priority packets will have a shorter latency while the low priority packets will be delayed. Minimizing the average message latency is not the goal of this policy.

3.5. A New Arbitration Policy: Look-Ahead

The *Look-Ahead* arbitration method proposed in this paper is intended to better distribute the load among the links, and hence, to improve the network throughput. Consider for instance the situation in Figure 2. The numbers in parenthesis represent the number of channels that contend for a particular physical link. We will refer to this number as the *link load*. There are two worms that want to occupy link 1, four worms that want to go on link 2, but no worm for link 3. Therefore, link 3 of switch 2 is unutilized. To maximize the throughput, it is better for switch 1 to try to send a flit which will use link 3 of switch 2. Such a flit, rather than waiting for others in front of him, can immediately proceed on link 3. To make this possible switch 1 must periodically "look-ahead" and collect information about the link loads of switch 2. The link load bits can be easily passed back together with the credit flow information. It must be noted that this *load information* is regarded by the upstream switch as a hint and does not oblige it to strictly take it into account. The ties in arbitration are broken by using the FCFS policy as a backup.

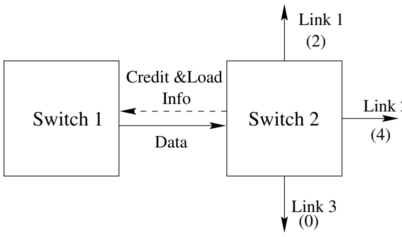

Figure 2. Look-Ahead operation

One possible source of overhead in the Look-Ahead scheme is the fact that the arbiter must know the next two hops of a worm. Thus, the routing table used to determine the next switch to be visited must be extended to also contain the identity of the second next switch (like in the Spider chip). This extension is not expected to affect the routing time as the two destinations are read in parallel.

Determining the link load threshold. A first question that arises is under what circumstances a link can be considered lightly loaded. What we need is a threshold value: if the link load exceeds this threshold, then the link is considered to be heavily loaded; otherwise it is lightly loaded. We performed experiments with synthetic workload to determine the optimum value for the threshold. Our results for two different packet sizes are presented in Figure 3. If we look at any curve on the graphs we see that, for good

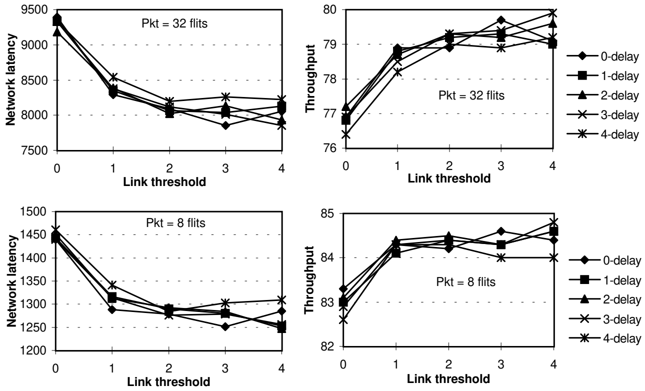

Figure 3. Look-Ahead arbitration method (above: Pkt = 32 flits, below: Pkt = 8 flits)

performance, the threshold should be 2 or greater. As we shall see shortly, to minimize the hardware implementation overhead, a value of 2 is preferred. This value will be used throughout the rest of the paper.

Look-ahead delay. A second implementation issue refers to the inherent time span between the moment of inspecting the link load (by the downstream switch) and the instant the upstream switch can effectively make use of this information. We will refer to this delay as *look-ahead delay*. Suppose that during cycle i the downstream switch collects load information. Then, this information can be passed back in cycle $i+1$, and the arbiter from the upstream switch can use it in cycle $i+2$. Hence, there can be a delay of at most two cycles. In some situations, if the switch is fast enough, the first two steps (load info gathering and transmission) can be combined into one, and the look-ahead delay is reduced to just one cycle. We performed synthetic workload simulations to see how this delay affects the performance of the switch. As Figure 3 shows, the distance between the curves named *x-delay* is very small which suggests that the impact of the look-ahead delay is minimal. The reason is that the link load tends to change very slowly in time. Thus, we can relax the requirements of our switch and use a delay of two cycles. This value will be used in all of our experiments.

Reducing the number of lines for the Look-Ahead policy Another concern refers to the way the load information is encoded. Of course we want as few bits as possible to minimize the number of lines between two adjacent switches. For an $n \times n$ switch we need to convey information about the load of only $n-1$ links. This is due to the fact that a worm cannot leave the switch using the same port it used for entering. Based on the previous results from Figure 3 we decided to have a link load threshold of 2. With just two bits we can encode four load levels: 0, 1, 2 and more than 2. Hence, for a 2D mesh the total number of bits needed for carrying load information is 8. The number of additional lines between the switches is given by the ratio between the number of bits used for link loads and the number of phits per flit. For example, we use 4 phits per flit, so the number of required lines is $8/4 = 2$. This number can be further reduced by half using one of the following three techniques.

1. Transmit only partial information about the load. In this case we send a bit mask specifying which links have the lowest load under the given threshold. The disadvantage is that there might be no worms that want to occupy one of these very lightly loaded links while there might exist some worms going to moderately loaded links. We will refer to this method as LA1.

2. Transmit 'vague' information about the load. In this variant we send a bit mask that shows which links have the load below the threshold and which are above. The information is vague in the sense that there is no distinction between the links whose load is below the threshold. Hence, the arbiter from the upstream switch might not choose the optimum one. This alternative will be referred to as LA2.

3. Transmit full load info, serially, across two flit cycles. The rationale behind this method is that the load cannot change sharply from one cycle to another. Even if we send information that is not current, its accuracy is good enough to provide good results. By using two cycles we basically increase the look-ahead delay. As Figure 3 suggests, this approach, called here LA3, is expected to perform very well.

Link no	Link load
1	0
2	4
3	2
4	2

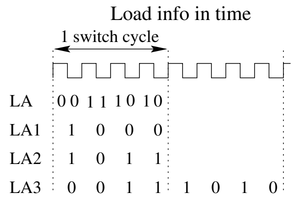

Figure 4. Examples of link load info used in three variations of the LA arbitration method

For a better understanding, Figure 4 shows an example of the link load information passed between the switches for all the variations of the Look-Ahead method introduced above.

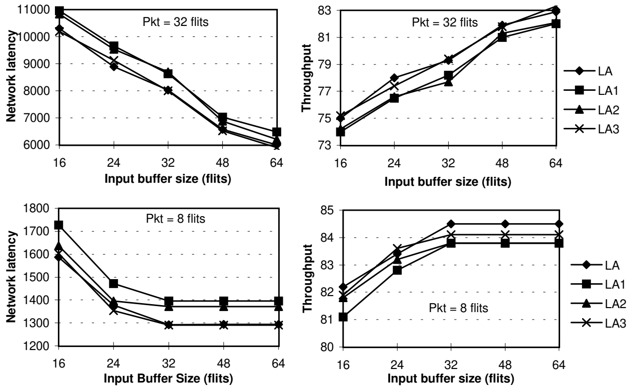

Figure 5. Look-Ahead arbitration variations

We evaluated these three variations of the Look-Ahead policy for several input buffer sizes. The results are plotted in Figure 5. On the graphs, LA denominates the original Look-Ahead method as described at the beginning of this section. LA1, LA2, LA3 represent the three look-ahead variations in the order they have been introduced. As we can see, using just partial (LA1) or vague (LA2) information about the link load does not degrade the performance too much (6-8% higher latencies and 0.5-1.5% lower throughput). However, LA3 is even a better alternative as its curve is almost identical with that of LA. This is a direct consequence of the fact that the look-ahead delay affects minimally the performance of LA.

4. Comparison of the Arbitration Policies

The goal of this section is two fold. First, it tries to determine the impact of arbitration on switch performance and to find out the circumstances under which the role of arbitration becomes more prominent. Second, it tries to rank the arbitration policies and to show the provisions that must be met for better performance. For this task we will use both synthetic workload and execution driven simulations.

The policies under examination are:

- First Come First Served (FCFS)
- First Come First Served - Keep Flow (FCFS-KF)
- Round Robin - Keep Flow (RR-KF)
- Shortest Message First (SMF)
- Look-Ahead (LA)

Due to the high latency, the simple Round Robin arbitration policy has been excluded from our results (only Round Robin - Keep Flow is presented). For Look-Ahead we consider a load threshold of 2 and a look-ahead delay of 2 cycles; also, full information (8 bits) about the load is communicated between the switches in just one flit cycle.

4.1. Results for Synthetic Workload Simulations

In Figure 6 (a) and (b) we plot the network latency and throughput of the five policies for packets of 32 flit and 8 flit respectively and for different input buffer sizes. As we can see the arbitration policy is an important factor in the design of a switch. The differences between the worst (RR-KF) and the best (LA) policy can be as high as 35% in network latency and 10% in throughput. These discrepancies become more important as the input buffer is smaller or the messages are larger. The reason is that for small buffers and large packets the network is more congested and the arbitration policy plays a larger role.

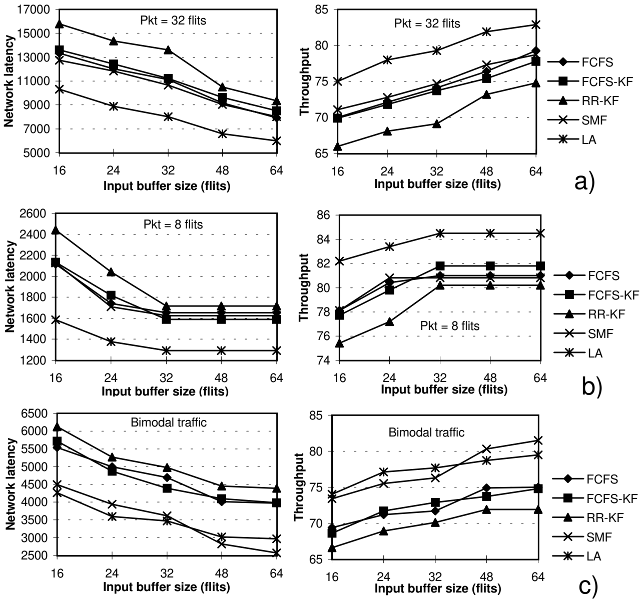

Figure 6. Comparison of arbitration policies

Between FCFS and FCFS-KF there are only minor differences. Of all arbitration policies, RR-KF has the worst performance. This is not surprising. It is already known that FCFS behaves better that RR because it tends to send flits from channels that are full. SMF policy behaves only slightly better than FCFS. Because packets are of same size there are fewer opportunities for SMF to select between various worms. In our experiments, SMF uses FCFS to break a tie. This is the reason why SMF is almost as good as FCFS. The Look-Ahead method clearly outperforms all the others in both network latency and throughput.

So far, we presented experimental results where all the packets had a fixed size of either 8 or 32 flits. For shared memory multiprocessor machines the network traffic is usually composed of two types of packets: read/write memory requests which are short, and reply messages which are often longer because they contain a whole cache line. Therefore, it is interesting to study the behavior of the network when loaded with a mixture of small and large packets. The small packets we use have 8 flits and they account for 50% of the input traffic. The remaining 50% of the traffic is formed by packets of 32 flits in length. Hence, the small packets outnumber the large packets by a factor of 4:1. The input buffer size is varied between 16 and 64 slots. The results are plotted in Figure 6 (c). Under bimodal traffic the SMF method improves considerably. As explained in Section 3.3 by giving priority to short messages we accelerate the process of freeing virtual channels, thus limiting the number of stalls due to unavailable VCs. With small buffers, usually, the switch runs out of buffers before it runs out of available channels. However, for large buffers the main impediment is the lack of free VCs. This is why SMF behaves better for large input buffer sizes, outperforming even the Look-Ahead policy. The ranking between all the other arbitration policies under analysis is maintained. LA method exhibits a latency reduction of 26% and a 6% improvement in network throughput over FCFS.

4.2. Results for Execution Driven Simulations

In this case the set of arbitration policies under analysis is slightly modified. Due to its similarity with FCFS we decided to drop the FCFS-KF scheme from the graphs. On the other hand we added two new policies. One is the Priority arbitration scheme described in Section 3.4. The other is a combination of Look-Ahead, Priority and SMF policies. The normalized execution time (with respect to FCFS) is presented in Figure 7.

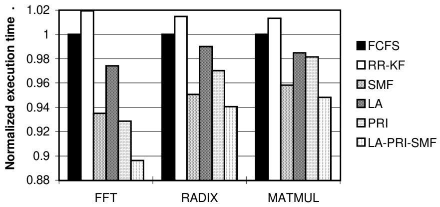

Figure 7. Effect of arbitration policies on execution time

In general, the ranking of the policies follows the pattern for bimodal traffic presented earlier in Figure 6 (c). However, the discrepancies between the policies are reduced because the network load is well beyond saturation (20-30% on average). Still, we can say that even in this situation the arbitration method cannot be neglected: it could lead to a reduction of about 12% in execution time.

The SMF policy proves to be a very effective method. There are two reasons for the good performance of SMF when compared to LA. First, the packet sizes (2 and 10 flits) are small when compared to the input buffer size (32 flits). We already showed that in such circumstances SMF outperforms LA. Second, SMF is more oriented towards latency reduction while LA strives to conserve bandwidth by a better distribution of the link loads. This makes LA an attractive policy when the network is very congested. Such conditions may arise for instance in database applications where the memories are stressed with a large number of requests.

The Priority arbitration policy, even though simple in implementation, achieves a relative good improvement in FFT (7% over FCFS, 9% over RR-KF) and a moderate improvement in MATMUL and RADIX. In MATMUL the number of read operations exceeds by far the number of write operations, so the priority mechanism does not have enough chances to be applied. In RADIX there are plenty of writes, but they are clustered together. For a good improvement we need the read and write operations to be as interspersed as possible.

Because all these three policies, LA, Priority, SMF complement each other we combined them in one powerful method: LA-PRI-SMF. This new policy is expected to perform as good as any of its components in any situation. Indeed, in our experiments it succeeds to outperform all the other policies for all the applications considered.

5. Enhancing the Arbitration Process by Virtual Channel Reservation

In a previous paper [11] we showed that reserving a few flits per virtual channel has a beneficial effect on performance because it eliminates the possibility of one worm occupying all the available slots and blocking the other worms. In this section we will show how a similar reservation idea can be applied with success at virtual channel level.

5.1. The Effect of VC Reservation on Uniformly Destined Traffic

In a traditional implementation of virtual channels, packets going to the same output can occupy more than one, or even all of the virtual channels belonging to an input port. In the latter case, even if there is space available in the shared buffer, these packets can block the way of others packets going to different outputs, because all the virtual channels from a particular input port are used up. Due to the pseudo-random nature of the input traffic, such situations can temporarily appear for a short time at different nodes in the mesh worsening the network latency and throughput. To avoid such a case, we propose to restrict the number of channels that packets going to a particular destination can use. The rest of the channels are said to be *reserved*. For instance, when one VC is reserved, no more than 3 channels can be used by worms going to the same destination (we assume 4 VCs per input port).

So far we have been elusive as to what destination means. It could be the output link to be used in the current switch, or it could mean the final node to be reached by the message. Our experiments (not presented here) indicate that the latter variant is better.

We enhanced the VC reservation process by observing that the number of reserved VCs should not be the same across the whole network. This number should vary according to the position of the link and of the switch in the mesh. For a better understanding lets look at an example where a worm moves from node 4 to node 0 (see Figure 1). Given the X-Y routing algorithm, the worm cannot take any turns because is already moving on the Y direction. Also, on Y direction, node 0 is the last one that can be visited. Therefore, the link between node 4 and node 0 can be used only by worms that have node 0 as their final destination. In this case, reserving any VCs would be totally inappropriate because those reserved VCs will never be used. Starting from this observation, for any given link we must determine the number of nodes that can be reached by worms going on that link. For a deterministic routing algorithm this number is fixed, and upon the placement of the switch in the mesh it can be easily computed and programmed into the router. Still, we have to determine the equation that gives us the number of reserved VCs per input port. We experimented with several choices and empirically found the following formula to perform the best:

$$adj\_\#VCres = min(2 * \#possible\_dest, \#VCres)$$

where $\#VCres$ is what we try to reserve and $adj\_\#VCres$ is the adjusted number based on the position in the mesh. In plain words, the above equation says that for most of the input ports we should try to reserve $\#VCres$, but for some nodes this number should be reduced so that no more than two VCs are reserved for every reachable node. We will refer to this scheme of reservation as *enhanced VC reservation* while the previous one where no adjustment is made will be denoted as *simple VC reservation*.

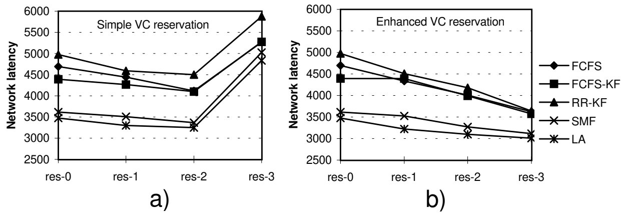

Figure 8. Effect of simple VC reservation (left) and enhanced VC reservation (right) on different arbitration policies for bimodal traffic

The benefits for the two reservation methods, simple and enhanced, are presented in Figure 8 (a) and (b) respectively. With simple reservation the number of channels that must be reserved is two. However, the enhanced reservation scheme is capable of better results, and in this case the optimum number of VCs to be reserved ($\#VCres$) is three. A second conclusion is that VC reservation can successfully be applied in conjunction with any arbitration policy. As expected, the arbitration methods that perform poorer benefit more from reservation. Still, the Look-Ahead method remains the best of all. The improvement in network latency is about 14-22% over the situation where no VC reservation is done. The next section will demonstrate that VC reservation plays an even more important role in case of a hot-spot situation.

5.2. Dealing with Hot-Spot Traffic

In the real world we can not expect the communication pattern of applications to be always perfectly balanced. Hot spots are often observed when processors contend for synchronization variables. In such cases, the buffers of the switches on the path leading to the hot-spot node are saturated with packets and block both hot-spot and regular packets alike. By restricting the number of virtual channels that packets going to one destination can take, we guarantee paths for non-hot packets. Therefore, regular traffic can pass through the network unhindered by the hot traffic.

The goal of this section is two fold: first, we want to show the impact of hot-spot on performance relative to various arbitration policies. Second, we are interested in seeing to what extent the VC reservation idea can prevent the negative effects introduced by the hot-spot. Figure 9 illustrates the variation of the network latency for different arbitration policies, when the input buffer has 32 slots. 90% of the packets are uniformly destined (destination is randomly selected) and 10% go to the hot-spot node (node 4 in our case). For reference, the situation when no hot-spot exists is also presented (NoHS).

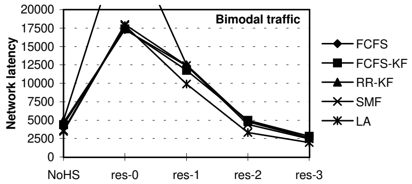

Figure 9. Negative effects of the hot-spot traffic and the effect of enhanced VC reservation on the uniformly destined traffic

Under hot-spot conditions, messages that go to the hot-spot node are substantially delayed. This effect is inevitable. Unfortunately, the hot-spot traffic repercuss on the uniformly destined traffic which is also delayed a great deal. None of the arbitration policies (including LA) is able to limit this negative effect. However, after applying the VC reservation technique the latency of the uniformly destined traffic improves considerably. As we reserve more virtual channels, the latency of UD messages decreases gradually, reaching levels even lower than the case where there is no hot-spot. The explanation for such a more than expected improvement is that, for short periods of time, even UD messages can create small hot-spots. The reservation process not only prevents HS traffic from interfering with UD traffic, but also helps to a better distribution of uniformly destined messages. The ranking between various arbitration policies is maintained even in the presence of the hot-spot.

6. Conclusion

In this paper we used synthetic workload and execution driven simulations to examine the behavior of a broad spectrum of link arbitration policies in the context of switch design. Our results show that link arbitration does play a significant role in the performance of an interconnect and this role is expected to increase as the switches become more complex. The discrepancies between the various arbitration policies can be as high as 35% in network latency, 10% in throughput and 12% in execution time of parallel applications. The arbitration becomes more important when the input buffer is smaller or when the messages are bigger. The Look-Ahead policy proposed in this paper proved to be very effective at high traffic rates. At a cost of just one additional wire between switches it provides significant improvements over traditional arbitration policies. The SMF policy, even though ignored in practice, has a great potential in situations of bimodal traffic and large input buffers. For shared memory multiprocessors the simple priority arbitration scheme introduced in this paper can significantly reduce the execution time for certain applications. We also showed that these arbitration methods can be successfully combined in even more powerful policies, applicable to a wide range of situations. However, none of the policies is able to prevent the negative effects of the hot-spot and therefore a supplemental mechanism is needed. Our solution of reserving a certain number of virtual channels depending on the switch position in the mesh can be added on top of any arbitration policy and is very efficient. Moreover, the VC reservation provides improvements even when there is no explicit hot-spot.

References

[1] L. N. Bhuyan. Analysis of interconnection networks with different arbiter designs. *Journal of Parallel and Distributed Computing*, 9(4):384–403, 1987.

[2] J. Carbonaro and F. Verhoom. Cavallino: The teraflops router and NIC. In *Proceedings of Symposium on High Performance Interconnects*, pages 157–160, August 1996.

[3] W. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[4] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. on Comp.*, C-36, 5:547–553, 1987.

[5] J. Ding and L. N. Bhuyan. Evaluation of multi-queue buffered multistage interconnection networks under uniform and non-uniform traffic patterns. *International Journal of Systems Science*, 28(11):1115–1128, 1997.

[6] M. Galles. Spider: A high-speed network interconnect. *IEEE Micro*, pages 34–39, January/Febrary 1997.

[7] F. E. Guibaly. Design and analysis of arbitration protocols. *IEEE trans. on comp.*, C-38, 2:161–171, 1989.

[8] M. Katevenis, D. Serpanos, and E. Spyridakis. Credit-flow-controlled ATM for MP interconnection: the ATLAS I single-chip ATM switch. In *Proceedings of the Fourth International Symposium on High-Performance Computer Architecture*, pages 47–56, January 1998.

[9] M. Katevenis, D. Serpanos, and P. Vatsolaki. ATLAS I: a General-Purpose, Single-Chip ATM Switch with Credit-Based Flow Control. In *Proceedings of the Hot Interconnects IV Symposium*, pages 63–73, 1996.

[10] D. J. Kinniment and J. V. Woods. Synchronization and arbitration circuits in digital systems. *IEE Proceedings, Computers and Digital Techniques*, 123(10):961–966, Oct. 1976.

[11] N. Ni, M. Pirvu, and L. Bhuyan. Circular buffered switch design with wormhole routing and virtual channels. In *Proceedings of the 1998 IEEE International Conference on Computer Design*, pages 466–473, October 1998.

[12] V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. In *Proceedings of the Third Workshop on Computer Architecture Education*, February 1997.

[13] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 programs: Characteriation and methodological considerations. In *Proceedings of the 22nd ISCA*, pages 24–37, June 1995.

[14] Q. Yang. Effects of arbitration protocols on the performance of multiple-bus multiprocessors. In *Proceedings of the 1991 ICPP*, pages I–600–I–603, Aug. 1991.