Exploring the Switch Design Space in a CC-NUMA Multiprocessor Environment *

Marius Pirvu, Nan Ni and Laxmi Bhuyan
Department of Computer Science
Texas A&M University
College Station, TX 77843
E-mail: {pirvum, ninan, bhuyan}@cs.tamu.edu

Abstract

The switch design for interconnection networks plays an important role in the overall performance of multiprocessors and computer networks. It is therefore crucial to study various factors in the switch design space and their influence on the system performance. In this paper we first propose a 4-D framework for the design of input queuing switches with wormhole routing and virtual channels. Then we explore the design space to examine in detail the impact of four parameters: virtual channel allocation, intra-switch connectivity, buffer space allocation and link arbitration policy. Our simulations, performed with an execution driven simulator with ILP processors, show that the cumulative effect of the four switch enhancements ranges between 7% and 38%. The most important parameter proves to be VC allocation method (up to 28% improvements in execution time). The other three bring about the same level of performance: between 1% and 7% depending on the application.

1 Introduction

CC-NUMA multiprocessors are the preferred choice of architects for scalable parallel computing. Examples include SGI Origin, Convex Exemplar and Sequent NUMA-Q machines. One of the factors that affect the performance of such a machine is the interconnection network. Major improvements in microprocessor design together with advances in process technology have led to exponential development of the processor speed. Techniques like data prefetching, release consistency memory models, non-blocking reads and out-of-order execution are all conducive to a higher rate of injecting messages into the network. Consequently, the network becomes more congested, and building high performance switches which can effectively deal with this level of congestion is a mandatory task.

In this paper we study the impact of four switch design parameters on application performance. The four parameters under investigation are: virtual channel (VC) allocation method, buffer space allocation, intra-switch connectivity and link arbitration.

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

VC allocation can be done statically or dynamically. In static VC allocation a packet going to a particular output link can be hosted by a single, predetermined VC. In dynamic allocation this restriction is removed and we can encounter situations where several VCs are occupied by worms going to the same link.

Buffer space allocation refers to the way the input buffer space is partitioned among virtual channels. At one extreme the buffer is equally split among the virtual channels and each VC has exclusive access to its own portion. On the other extreme the buffer space is entirely shared by the virtual channels. In this case the VCs must dynamically allocate buffer slots from a common pool whenever needed. Of course, we can have a compromise where part of the buffer is dedicated to each VC and part is shared.

Intra-switch connectivity refers to the number of connections between the group of virtual channels from an input port and the crossbar inside the switch. More connections means more chances to send several flits out in the same clock cycle and better utilization of the output links.

Link arbitration is used to decide which flit will use a particular physical link next cycle. We test a wide variety of arbitration policies to determine which is more suitable for parallel applications.

There exists a plethora of papers on switch design [4, 10, 5, 16], but the vast majority of these use synthetic workload for the purpose of experimental evaluation. Synthetic generated workloads are sometimes preferred because the experiments can be carried out much faster than using an execution driven simulator. Thus, we can afford to investigate a broad spectrum of parameters and rapidly filter out the design choices that are not interesting or even harmful. However, relying entirely on synthetic workload simulation can sometimes be misleading. The traffic pattern generated by a real application is very complex and cannot be accurately emulated by synthetically generated workload. Applications usually have phases, and each phase might exhibit a different communication pattern. Periods of extensive computing can alternate with periods of high communication needs which might create bursts of messages or even hot-spots within the interconnect. Also, cache invalidation and synchronization patterns are very difficult to be captured by synthetic workloads. For this reason we believe that execution driven simulations are a necessary step in validating new switch design ideas.

There are only a few papers in the literature that investigate the impact of switch design on application performance. Kumar and Bhuyan [9] were the first to study the performance of applications on wormhole routed networks with virtual channels. They limit themselves to only three network parameters: number of VCs, number of flit buffers per channel and number of links between the compute node and the router. Their results show that four VCs per input port offers the best performance in most cases and that 2 to 4 flit buffers per channel are usually enough. Vaidya et al. [18] analyze the performance benefits of virtual channels and adaptive routing in 2-D meshes. They find that these enhancements reduce the network latency to varying degrees depending on the application and that the improvement on execution time is most of the time negligible. They conclude that the raw latency of a switch is more important than improvements meant to cut down the network contention and reduce blocking time of messages. Bhuyan et al. [1] compare four different switch architectures in MINs. The switches are different in switching technique, buffer length and virtual channels. The authors break down the message latency per stages and show that, for large messages (replies carrying cache line data), the delay at the network interface (stage 0) is by far the most important. They trace this aspect to the burstiness nature of the communication pattern determined by both the application behavior and memory data placement.

We do not want to repeat the experiments carried out by these papers, but rather to address design decisions which usually received little attention from the researchers: buffer space allocation, VC allocation, intra-switch connectivity and link arbitration. Our goal is to determine to what extent such factors contribute to the message latency and, more importantly, to the execution time of parallel applications.

The three previous studies mentioned above assumed a simple scalar processor model with blocking reads. This model inherently limits the rate at which messages are generated and injected into the network. We believe that this is a severe restriction which does not reflect the current processor technology and therefore might underestimate the impact of network congestion on application execution time. In contrast, our simulations are based on a detailed execution driven simulator with ILP processors, RSIM [13]. We simulate state-of-the-art superscalar processors with branch prediction, out-of-order speculative execution, non-blocking reads and release consistency shared memory model. Analyzing the difference between a simulator that assumes a simple processor model and RSIM we find large discrepancies.

To summarize, the paper makes the following original contributions:

- Proposes a 4-D framework for input queuing switch design that gives the taxonomy of the design space.
- Quantifies the impact of four switch design choices on parallel application performance.
- Identifies which of the four choices is of more importance and under what circumstances.
- The analysis is carried out using a detailed execution driven simulator with ILP processors.

The rest of the paper is organized as follows: Section 2 describes the four switch design choices that make the subject of this paper. Section 3 describes our test-bed and other architectural assumptions. The experimental results are analyzed in Section 4. Finally, section 5 concludes the paper.

2 A 4-D Framework for Input Queuing Switch Design

A $k \times k$ switch should be able to receive messages from k input ports and route them to k output ports with minimum delay. If conflicts at the output ports occur, those packets that cannot pass should be buffered and transmitted later.

A straightforward design is to provide each input port with a first-in-first-out (FIFO) buffer. However, if the first packet in a buffer is blocked because of a conflict at the output port, other packets behind it cannot be transmitted even if their intended output ports are idle. A common solution to this *head of line* (HOL) problem is the implementation of virtual channels [4] within the input buffer. Still important design issues remain. Specifically the switch architects must decide:

- How to allocate a VC for an incoming packet
- How to partition the buffer space among the VCs
- How to connect the multiple VCs from an input port to the crossbar
- How to pick the channels that will use the output links next cycle

Fortunately, these issues are more or less orthogonal to one another so we can deal with them separately. This allows us to build a four dimensional model for the input queuing switch design space. We will classify the design space according to the above four indices, namely virtual channel allocation method, buffer space allocation scheme, intra-switch connectivity to the crossbar and link arbitration policy. For better understanding, we first explore the design space based on the first three indices, which we refer to as input buffer management, and then add the fourth index (link arbitration) on top of the 3-D model.

2.1 3-D Model for Input Queuing Switch Design

Virtual channel allocation: Upon the arrival of a message at a switch, we have two ways to assign it to a virtual channel. In the *static* scheme, there is only one potential channel that a message can be allocated to. The channel number is pre-determined based on the destination of the message. Packets to the same output port will be given the same channel number. Other channels can not be used even if they are free. The *dynamic* allocation scheme, on the other hand, does not have this restriction. Regardless of its destination, an incoming packet can occupy any virtual channel available. Obviously, static allocation has poor virtual channel utilization. Moreover, the number of virtual channels associated with an input must be equal to the number of the output ports. In contrast, the dynamic allocation scheme has better virtual channel utilization and a simpler routing mechanism. The number of virtual channels in each input buffer does not depend on the number of output ports. However, one potential problem with the dynamic allocation scheme is that, since a channel can be occupied by messages to any output, it is possible that all the virtual channels are hogged by packets going to the same output port. Consequently, it may obstruct the way of other messages. To avoid this, a virtual channel restriction scheme has been proposed in [14] that prevents messages going to the same link from occupying all the virtual channels.

Buffer space allocation: The buffer space in each input port can be allocated in two ways. Either the buffer is divided equally among the virtual channels or the entire buffer is shared by the channels as demanded. A *separate buffer* approach can be implemented as multiple dedicated small FIFOs corresponding to the virtual channels of an input block. The disadvantage is that the buffer for one small queue can be full while all the other small queues of the same input port can be empty. In contrast, a *combined buffer* approach provides more flexibility and better utilization of the buffer. The combined buffers are commonly implemented using linked lists. The circular buffer approach we proposed in [12] has simplified the hardware design.

Intra switch connectivity to the crossbar: From each input of a switch, there can be either one or more paths connected to the crossbar, corresponding to *single connected* or *fully connected* switches respectively. In the former case, at most one flit from each input port is allowed to be transmitted to the output port at any given cycle. Packets to other destinations from the same input port have to wait even though their corresponding output ports are idle. With full connection, packets to different output ports could pass through the crossbar simultaneously as long as conflicts with packets from other inputs are resolved. Switches with single crossbar connection have less complexity. The input block needs only one write port from the input link and one read port to the crossbar input. The arbitration process for a single connection switch usually takes two steps. First, one virtual channel from each input block is chosen to use the connection to the crossbar. These winners participate in the second step where contention for output links is resolved. Since the first step of arbitration is done by each input block independently, we can imagine a scenario where all the selected channels from different input blocks are to the same output port. In this case only one flit can be transmitted to an output port and the throughput is degraded. Fully connection works well in situations where the messages in various input ports are "unbalanced". In an extreme case, all virtual channels in a specific input port are occupied while no messages are present in other input ports. Under this situation, single connection allows only one flit to pass to the crossbar and the bandwidth is lost. With full connection, multiple flits can be passed given that they are bound for different output ports.

Taking into consideration these three switch parameters we can depict the design space of an input queuing switch in a 3-D fashion as in Figure 1. The VC allocation scheme

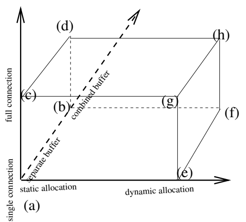

Figure 1. 3-D model for switch architecture

is on the X axis. The Y axis shows the intra-switch connectivity to the crossbar. The buffer space allocation scheme is shown on the Z axis. There are eight possible combinations:

a. **SASCSQ - Statically Allocated Singly Connected with Separate Queue** [17] (see Figure 2(a)).

b. **SASCCQ - Statically Allocated Singly Connected with Combined Queue** (see Figure 2(b)).

c. **SAFCSQ - Statically Allocated Fully Connected with Separate Queue** (see Figure 2(c)).

d. **SAFCCQ - Statically Allocated Fully Connected with Combined Queue** [5] (see Figure 2(d)).

e. **DASCSQ - Dynamically Allocated Singly Connected with Separate Queue** (see Figure 2(e)).

f. **DASCCQ - Dynamically Allocated Singly Connected with Combined Queue** [17] (see Figure 2(f)).

g. **DAFCSQ - Dynamically Allocated Fully Connected with Separate Queue** [11] (see Figure 2(g)).

h. **DAFCCQ - Dynamically Allocated Fully Connected with Combined Queue** [12] (see Figure 2(h)).

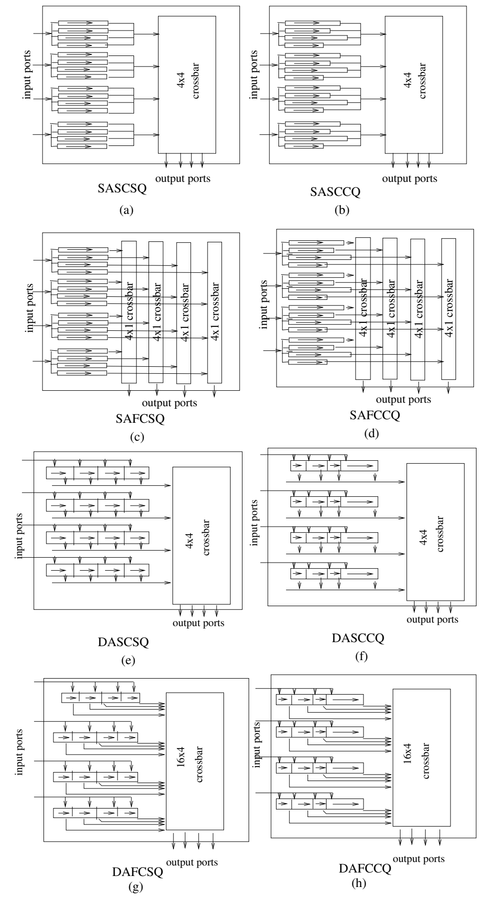

Figure 2. Examples of switch architectures

In addition to the three factors discussed, another index in the design space is crossbar arbitration policy. We will discuss this factor next.

2.2 Arbitration Policies for Shared Memory Operations

In the following we briefly describe the link arbitration policies used in this evaluation.

Round Robin One of the most popular arbitration policies is *Round Robin*. However, the disadvantage of this method is that the network latency tends to be uniformly increased for all the messages. To remedy this drawback the arbiter should give priority to the message that is currently flowing out of the switch. When this message has drained out completely, the simple Round Robin policy is resumed. We will refer to this scheme as *Round Robin - Keep Flow* (RR-KF). A slightly modified variation of this scheme is employed in the routers for the Cray T3E network [15].

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. 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.

Shortest Message First From the process scheduling theory we know 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. In a shared memory machine the short packets are constituted by the memory requests and coherence messages (invalidation requests and acknowledgments for example).

A second advantage of the SMF policy is increased VC availability. The traditional implementation of VCs allows only one packet per VC. Even if there are free buffers, packets can block because all VCs are busy. SMF relieves this drawback by freeing VCs as fast as possible. 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.

Priority Based In this arbitration policy each packet is assigned a priority level. Packets with higher priority are always selected first. When multiple packets have the same priority, the FCFS method is used to break the tie.

The basic idea of this arbitration scheme is to assign a higher priority to requests that are critical [14]. In a CC-NUMA multiprocessor that employs release memory consistency models the latency of write operations can be hidden to a large extent. On the other side, read operations are more critical because the processors will soon stall in the absence of the desired data. Therefore, it is 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.

Look-Ahead Arbitration The *Look-Ahead* (LA) arbitration method [14] is intended to better distribute the load among the links, and hence, to improve the network throughput and latency. The idea is to try to send flits that will occupy unused links in the next router. To make this possible the upstream switch must periodically "look-ahead" and collect information about the link loads of the downstream switch. The link load is defined as the number of channels that contend for a particular physical link. To minimize the hardware overhead this information is encoded with just 2 bits. Thus, we can express 4 levels of load: 0, 1, 2 and more that 2. Links that have a load of more than 2 are considered as heavily loaded and from that perspective they fall into the same category. The link load bits can be easily passed back together with the credit flow information.

3 Simulation Environment

Our simulator is based on RSIM [13] which accurately measures the contention at all subsystems. However, we completely changed the network part to allow for a detailed cycle by cycle simulation of the interconnect. The architecture we simulate is a shared memory machine with 16 nodes connected by a 4×4 mesh like in Figure 3(a). Each node is composed of a processor, two levels of cache, a memory module, a network interface and a switch (see Figure 3 (b)).

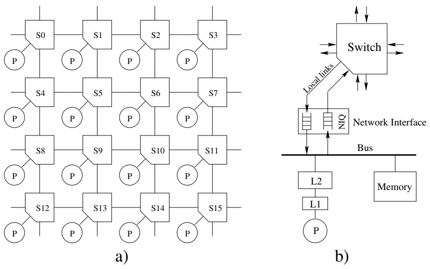

Figure 3. Simulated architecture

We simulate a state-of-the-art, 800 MHz, 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 modules are 4-way interleaved and are distributed across the system in a round robin fashion at page granularity. The size of a page is 4 KB and the latency to read a memory block is 48 processor cycles. The system bus is a 200 MHz split transaction bus with a width of 128 bits. 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.

The switch at each node 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 uses wormhole routing and has four VCs at each input port. The physical links between switches are considered to operate at processor fre-

Switch parameters		Cache parameters	
flit size	8 bytes	L1 size	16 K
link width	2 bytes	L1 assoc.	2 way
switch delay	12 pcycles	L1 latency	1 pcycle
wire delay	1 pcycle	L2 size	128K
routing delay	4 pcycles	L2 assoc.	4 way
routing alg.	X-Y	L2 latency	8 pcycles pipelined
VCs	4 per input	line size	64 bytes
switch cycle	4 pcycles	write policy	write back

Table 1. Default architectural parameters

quency, 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. The arbitration is performed at flit level, every switch cycle. The pin-to-pin flit delay is 3 switch cycles. One cycle is needed to read the flit from the reception buffers into the core, one is needed for arbitration and routing and another one is needed to send the flit to the transmission buffers. This model is similar to the one used in the Spider chip except that we eliminated the need for synchronization. The total size of the buffer at each input port is varied between 16 and 64 flits. The packets pushed into the network have two different sizes: 10 flits if the carry a cache line and only 2 flits otherwise.

As benchmarks we use six applications: FFT, RADIX, MATMUL, FWA, EM3D and NBF. FFT computes the 1-D Fast Fourier Transform of 2^{16} complex points. RADIX sorts 2^{18} integers using the radix sort algorithm. These two applications are taken from the SPLASH-2 benchmark suite [19]. MATMUL is a small kernel which does the multiplication of two 128x128 matrices. FWA computes the shortest path between any two points in a 128 node directed graph and is based on the well known Floyd-Warshall algorithm. EM3D [3] models the propagation of electromagnetic waves through objects in three dimension. It operates on a bipartite graph with 2048 nodes, 20% of which are local. NBF (the Non-Bonded-Force kernel) is a molecular dynamics simulation taken from the GROMOS benchmark [7]. Its computational structure resembles the non-bonded force calculation in CHARMM which is a well-known molecular dynamics code. The number of molecules is 16386.

Other relevant characteristics of the applications are shown in Table 2. To keep the simulation time within reasonable limits the problem sizes that we run are rather small.

Application Name	Read / Write mem. requests (x 1000)	Overall miss rate (percent)	Network utilization (percent)
FFT	64 / 48	1.75	25.0
RADIX	54 / 99	1.11	26.8
MATMUL	20 / 4	0.53	12.1
FWA	22 / 3	0.34	7.0
EM3D	267 / 40	13.14	49.2
NBF	162 / 92	2.22	9.1

Table 2. Characteristics of applications

4 Results

The goal of this section is to determine the impact of the four switch parameters on application performance and to find out the circumstances under which their role becomes more prominent.

4.1 Effect of Input Buffer Management

Figure 4 presents the network latency and normalized execution times for different applications. The factor that affects the most execution time of applications is the VC allocation method. When dynamic VC allocation is used we observe improvements ranging from 3% (NBF) to 28% (EM3D) (SASCSQ vs. DASCSQ). These improvements are consistent irrespective of the buffer size. The bad performance of statically allocated VC method is due to a poor VC utilization. This factor is amplified at the nodes found at the corners of the mesh which have a reduced number of neighbors. This implies a reduced number of possible destinations and therefore a reduced number of worms per input port. A similar situation arises when most of the worms travel in one direction (as in the case of a hot-spot): a worm is forced to wait for the previous worm to completely leave the VC before it can move to that input port. The switch delay also plays a role. For better understanding let's assume a hypothetical scenario where all messages enter a switch through port 0 and leave the switch through port 1. Because all the messages are destined to the same output port they will want to occupy the same VC in the switch. Hence, only one message can be present at any given time in the input buffer of port 0. Suppose that the last few flits from message A are currently in the input port. After message A drains completely out of the switch message B can enter. However, before the header of message B can go to the next switch, a time equal to switch delay must elapse. Meanwhile the output link 1 is not utilized. So, not only the latency is increased, but also the throughput becomes worse. These effects are expected to be lower when the switch delay is smaller. Indeed, when we decreased the switch delay to 1 switch cycle the advantage of having dynamically allocated VC shrank from 11% to 8% for FFT, from 12% to 10% for RADIX and from 25% to 16% for EM3D. The results reported above are for a single connected switch with separate queues (buffer size is 64 flits).

The gains of the buffer space allocation scheme are heavily dependent on the total buffer size. Using combined queues improves the execution time by 2-7% for small buffers (16 entries, SASCSQ vs. SASCCQ), but has no effect when buffers are fairly large (64 entries). Also, the improvement derived from combined queues reduces if the switch already employs dynamically allocated VCs. However, it is interesting to observe that small combined buffers are able to achieve the same level of performance as large split buffers.

Intra-switch connectivity plays a lesser role with respect to application performance especially when the buffers are small (3% for FFT, 2% for EM3D, negligible for others). Nevertheless, its role increases when VC are allocated dynamically or when input queues are either large or combined (3% for RADIX, 5% for EM3D). In such situations intra-switch connectivity becomes the second factor as importance.

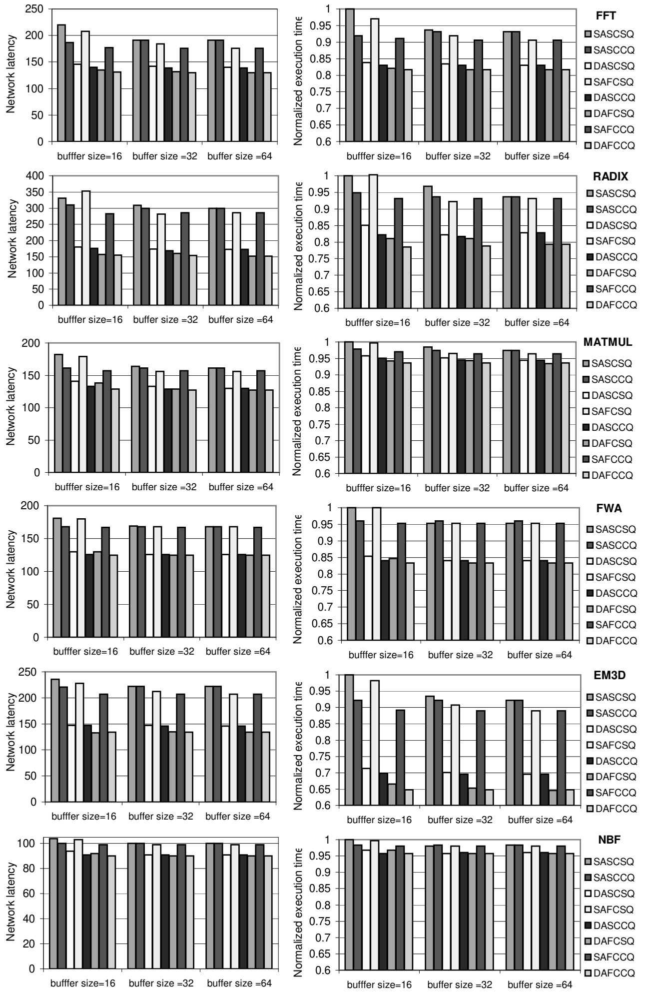

Figure 4. Impact of input buffer management on application performance

There is a good correlation between the average message latency and application execution time as one closely follows the trend of the other. However, the influence of message latency on execution time varies across the applications. For example a 40% reduction in message latency for EM3D (SASCSQ vs. DAFCCQ, 16 flit buffer) translates in a 35% improvement in execution time, while a huge 50% message latency reduction for RADIX corresponds to only 20% improvement in execution time. Of course this is a direct consequence of the nature of the application: applications that are more communication intensive benefit more from network improvements.

We mentioned in Section 1 that a simple simulator which assumes simple scalar processors with blocking reads might generate low levels of traffic and therefore underestimate the importance of switch parameters. To verify this hypothesis we forced our simulator to fetch only one instruction per cycle. Also, we disabled the ability to have several outstanding misses (a hit under miss is allowed though). We found that such a simple simulator is conducive to large

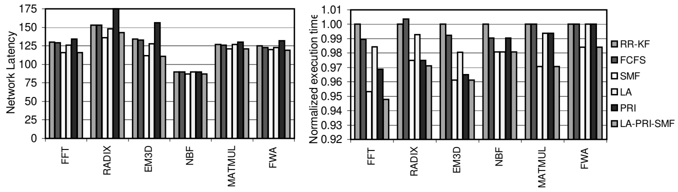

Figure 5. Impact of link arbitration on application performance

errors. For example, the EM3D execution time improvement of the DAFCCQ switch over the SASCSQ switch (16 flit buffers assumed) is only 7% with the simple simulator while with RSIM simulator is 35%. Similarly, the DAFCCQ switch network utilization is only 14.9% while with RSIM is 44.5%. Therefore, we see a huge gap between the results of the two simulators. The conclusion is that the processors utilized in CC-NUMA machines indirectly affect the level of performance provided by switch enhancements. As processors are capable to generate messages at a faster rate, support more outstanding misses or speculatively issue memory operations, the switch design becomes more important.

4.2 Effect of Link Arbitration

In this section we analyze the performance of the 5 arbitration policies described in Section 2.2. Our evaluation is performed on a DAFCCQ switch whose input buffer size is 32 flits. We selected this configuration so that the only source of improvement remains arbitration. For any given application the execution time is normalized to the performance of RR-KF (round-robin-keep-flow).

As we can see from Figure 5, arbitration alone can improve the execution time by as much as 5%. 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 turns out to be one of the best arbitration policies. The explanation resides in the fact that the buffer size (32 flits) is rather large when compared to the message sizes (2 and 10 flits). In this situation the messages usually stall not because of lack of buffers, but because of lack of free virtual channels. SMF accelerates the process of freeing virtual channels and thus generates fewer stalls.

LA policy shows a lesser improvement when compared to SMF. This is because 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 moderate improvement: about 3% in FFT, RADIX and EM3D. One interesting remark about priority based arbitration is that the average message latency is larger that in any other policy. The explanation is that priority based arbitration accelerates urgent messages while delaying the less critical packets (write operations mainly).

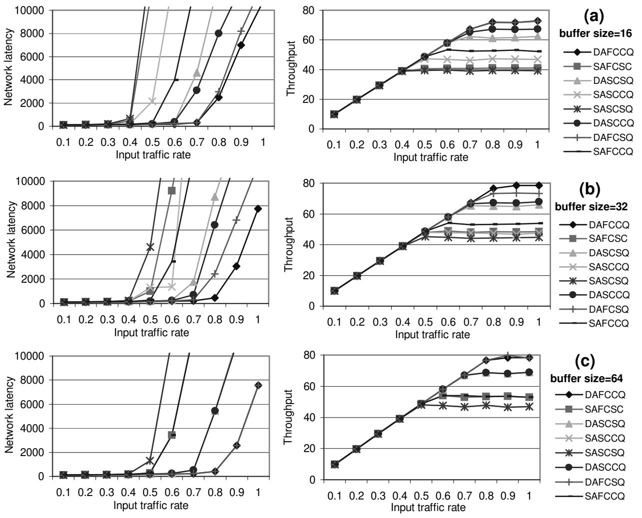

Figure 6. Impact of input buffer management on synthetic workload

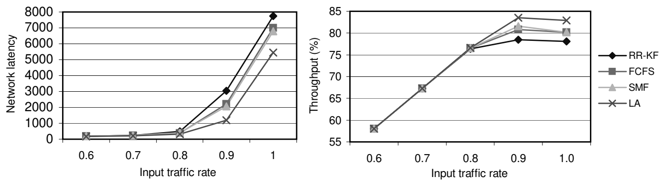

Figure 7. Impact of link arbitration on synthetic workload

Therefore, write messages experience a fairly large latency which is conducive to a high average network latency.

Because all these three policies, LA, Priority, SMF proved to be good in one respect or another 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.

When there are more worms in the network the arbitration is expected to have a greater impact on performance simply because for each link there are more options (more worms) to choose from. To verify this assertion we increased the number of miss status holding registers (MSHR) [8] at each cache from 8 to 64. Thus, each processor can now have up to eight times more outstanding misses. Our results (not presented here for lack of space) show that under these circumstances arbitration can improve the execution time by as much as 11%.

Arbitration can reduce only part of the blocking time of messages through a better scheduling of messages. However, it cannot affect the raw latency of a switch. Therefore, it is naturally to assume that, the shorter the switch delay, the better the relative improvement of the arbitration. This hypothesis is supported by our experiments. When the switch delay is reduced to just two network cycles (down from three) arbitration can reduce the execution time by an additional 1-2%.

4.3 Comparison with Synthetic Workload Simulations

In Section 1 we suggested that extrapolating the results of synthetic workload simulation to real life applications is a very difficult task. To verify this hypothesis, we examined the performance of various switch designs under a synthetically generated workload. In this case the destination of the packets is randomly determined and their length is 10 flits.

Our results depicted in Figures 6 and 7 show that, with few exceptions[1], the trends observed for synthetic workload simulations are usually similar to the ones for application simulations. At high input traffic rates (close to saturation), the discrepancies between various switch design alternatives are bigger than those revealed by execution driven simulations. On the other hand, for low input traffic rates these discrepancies completely disappear since the design issues explored in this paper are aimed only at alleviating the negative effects of network congestion. It follows that the synthetic workload simulations give us a good qualitative appraisal of the various switch parameters, but fail to give a quantitative one. We can estimate which architectural improvement is better, but not by how much. The main reason is because in synthetic workload simulations a fundamental parameter is the input traffic rate. We cannot establish an appropriate value for this parameter unless we examine each application. Even so the results can be misleading. For example the average input traffic rate generated by our applications varies between 10% and 50%. For these levels the synthetic workload simulations show no difference between various switch designs. Nevertheless, the execution driven simulations contradict this assertion. The explanation is simple: the traffic pattern of applications is usually bursty. While the average traffic rate is modest, there are moments when the network is flooded with messages. Depending on the weight of these intensive communication phases, switch design parameters can make a smaller or a greater impact on application performance. A second factor is the size of the messages. In CC-NUMA machines we deal with 3 types of messages: requests, which are usually short, replies which are usually large and coherence messages which can be either short or large. It is very hard to emulate the right mixture of messages in synthetic workload simulations. A third impediment is the non-uniformity of the traffic generated by applications. Such non-uniformities could create temporary hot-spots in different parts of the network, hot-spots which are very difficult to be captured by synthetic workload simulations.

4.4 Summary of Experiments

In this section we would like to summarize our experimental observations in the form of a set of recommendations.

Given that the switch under investigation follows the traditional implementation of virtual channels which allows a

[1]SMF arbitration policy does not perform well for synthetic workload due to the fact that all messages have same length. Also, LA arbitration policy performs much better in synthetic workload simulations than in execution driven simulations.

single message per VC, our results suggest that dynamically VC allocation is a must. It provides very good improvements and incurs little implementation complexity. If the messages to be transmitted vary widely in size or if their size is large, implementing combined queues is definitely worthwhile. The suitability of having a fully connected switch must be carefully judged from the perspective of additional switch complexity and the level of gains from the particular application to be used.

However, if the messages are small like in CC-NUMA multiprocessor machines, it makes more sense to build the input buffer as separate queues. This reduces the complexity to a large extent and also simplifies the implementation of fully connected switches because we do not need to have buffers with multiple read ports. Thus, it is probably convenient to add full connectivity to such a switch especially if the number of virtual channels is relatively small (4 to 6).

As far as arbitration is concerned we advise against round-robin policy. First-come-first-served is better and does not add complexity. For CC-NUMA multiprocessors priority based arbitration is a good option even though it requires support from other subsystems to set the priority of messages. In a highly congested network with random like traffic (a multiprogramming environment for example) Look-Ahead policy is a serious contender. However, shortest-message-first policy is probably the most versatile. Unless all the messages have the same size it provides good performance under a wide variety of traffic patterns. Nevertheless, we must provide an additional mechanism to preclude starvation.

5 Conclusion

In this paper we studied the impact of VC allocation method, buffer space allocation scheme, intra-switch connectivity and link arbitration on application performance. All of these switch parameters are meant to cut down the network latency by reducing the blocking time of packets. From this perspective they are useful only when the level of congestion in the interconnect is high. As a consequence, the effectiveness of these parameters on influencing application performance is directly related to application behavior. For applications that are communication intensive, like FFT, RADIX or EM3D, the cumulative effect of the four switch enhancements ranges between 20% and 38%. The most important parameter proves to be VC allocation method (up to 28% improvements in execution time). The other three bring about the same level of performance: between 1% and 7%. Buffer space allocation is more important when the size of the buffer is small. Also, its effectiveness seems to decrease when other enhancements are added to the switch. In contrast, intra-switch connectivity is more important when the switch has dynamically allocated virtual channels and the buffer space is either large or dynamically allocated. For the configuration considered link arbitration affects execution time in the range of 1% to 5% depending on the application. We showed that alternative arbitration policies like shortest message first, look-ahead or priority based are capable of outperforming the traditional round-robin and first-come-first-served policies.

As processors become faster and faster and allow more and more outstanding misses the level of congestion inside the interconnect is expected to increase. Consequently, the importance of these four switch parameters analyzed in this paper is also expected to grow.

References

[1] L. N. Bhuyan, H. Wang, R. Iyer, and A. Kumar. Impact of switch design on the application performance of cache-coherent multiprocessors. In *Proceedings of the IPPS/SPDP '98*, pages 466–475, March 1998.

[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] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. Eicken, and K. Yelick. Parrallel programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, Nov. 1993.

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

[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] W. Gunsteren and H. Berendsen. GROMOS: GROningen MOlecular Simulation software. Technical report, Laboratory of Physical Chemistry, University of Groningen, 1988.

[8] D. Kroft. Lockup-free instruction fetch/prefetch cache organization. In *Procs. of the 8th ISCA*, pages 81–87, May 1981.

[9] A. Kumar and L. N. Bhuyan. Evaluating virtual channels for cache-coherent shared memory multiprocessors. In *Procs. of ACM Int'l Conf. on Supercomputing*, May 1996.

[10] M. Kumar and J. Jump. Performance enhancement in buffered Delta networks using crossbar switches and multiple links. *Journal of Parallel and Distributed Computing*, 1(1):81–103, 1984.

[11] A. Mu, J. Larson, and R. Sastry. A 9.6 GigaByte/s throughput Plesiochronous routing chip. In *Proceedings of COMPCON'96*, pages 261–266, 1996.

[12] N. Ni, M. Pirvu, and L. Bhuyan. Circular buffered switch design with wormhole routing and virtual channels. In *Procs. of the 1998 IEEE Int'l Conf. on Computer Design*, pages 466–473, October 1998.

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

[14] M. Pirvu, L. Bhuyan, and N. Ni. The impact of link arbitration on switch performance. In *Proceedings of the Fifth HPCA*, pages 228–235, January 1999.

[15] S. Scott and G. Thorson. The Cray T3E network: Adaptive routing in a high performance 3D torus. In *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, pages 147–156, Aug. 1996.

[16] R. Sivaram, C. B. Stunkel, and D. K. Panda. HIPIQS: a High-performance switch architecture using input queueing. In *Proceedings of the IPPS/SPDP '98*, pages 134–143, March 1998.

[17] Y. Tamir and G. L. Frazier. Dynamically-allocated multi-queue buffers for VLSI communication switches. *IEEE Transactions on Computers*, 41(2):725–737, June 1992.

[18] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. Performance benefits of virtual channels and adaptive routing: An application-driven study. In *Procs. of the 11th Int'l Conf. on Supercomputing (ICS-97)*, pages 140–147, July 1997.

[19] 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.