

# Performance Benefits of Virtual Channels and Adaptive Routing: An Application-Driven Study\*

Aniruddha S. Vaidya

Anand Sivasubramaniam

Chita R. Das

Department of Computer Science and Engineering  
The Pennsylvania State University  
University Park, PA 16802  
E-mail: {vaidya, anand, das}@cse.psu.edu

## Abstract

Recent research on multiprocessor interconnection networks has primarily focussed on wormhole switching, virtual channel flow control and routing algorithms. These architectural features are aimed at enhancing the network performance by reducing the network latency, which in turn should improve the overall system performance. Many research results support this design philosophy by claiming significant reduction in average message latency. However, these conclusions are drawn using synthetic workloads that may not necessarily capture the behavior of real applications. In this paper, we have used parallel applications for a closer examination of the network behavior. In particular, the performance benefit from enhancing a 2-D mesh with virtual channels (VCs) and a routing algorithm (oblivious or fully adaptive) is examined with five shared memory applications using an execution-driven simulator, SPASM.

In order to analyze the performance implications in greater detail, we also consider other parameters that have a direct bearing on network traffic. These are the number of processors used to solve a problem, problem size and memory consistency model. Simulation results show that VCs can reduce the network latency to varying degrees depending on the application. Similar gain is possible with a fully adaptive routing algorithm compared to the oblivious routing. However, with respect to the overall execution time, the performance benefit using these enhancements is negligible. Moreover, this benefit is negated when we consider the cost of implementing the VCs. These results suggest that the performance rewards may not justify the cost of these enhancements. Rather, we need to emphasize on improving the raw network bandwidth by simpler and improved router designs.

## 1 Introduction

The complex interaction between a parallel architecture and an application makes it essential to use realistic workloads for evaluating parallel systems. Performance analyses of processors, caches, memory and I/O subsystems have therefore been conducted with

\*This research was supported in part by the National Science Foundation under grants MIP-9406984 and MIP-9634197.

parallel benchmarks. However, unlike the other subsystems, interconnection network design and analysis has rarely used the knowledge of workloads generated by parallel applications. In this paper, we study the performance of a two-dimensional mesh interconnection network under realistic workloads and analyze its impact on parallel application performance. Our focus, in particular, is on shared memory machines, which are fast becoming the platform of choice for scalable parallel computing. Our specific aims are to verify whether the promised performance improvement using recently proposed network enhancements, such as virtual channels (VC) and routing adaptivity, indeed occurs for real applications, and if so do these benefits override the cost of providing these enhancements.

There are two differing perspectives of viewing the multiprocessor interconnection network. From the viewpoint of a software designer or an application programmer, it helps to make certain simplifying assumptions about the interconnection network such as assuming a constant delay introduced by the interconnection network or a simple model, which does not take into account the details of message traversal within the actual network. These assumptions are sufficiently accurate when the objective is to minimize the communication required. By making these assumptions, performance evaluation of the system can be simplified and speeded-up. However, performance results obtained using these models maybe suspect when used to evaluate system/application performance with various parallel workloads [7].

Interconnection network designers have a more network-centric viewpoint. From this viewpoint, improving the network performance is critical. Network topology, switching mechanism, routing, flow control, and communication workload, together determine the network performance. Until recently, network research has primarily focussed on the first four parameters to optimize network latency and throughput. Research results show that low-dimensional networks such as  $k$ -ary  $n$ -cubes and meshes with wormhole switching and virtual channel flow control can provide the needed performance [12]. In addition, adaptivity in message routing can boost the network performance further [15, 18, 16, 13, 9, 5, 19, 4]. Many of these research ideas have found their way into contemporary commercial systems. For example, the Cray T3E router [21], the Intel Cavallino router [8], and the SGI SPIDER routing chip [17] are routers for direct networks and use wormhole switching, and virtual channel flow control. All of these routers use four VCs and oblivious routing, while the T3E router has adaptive routing capability.

Most performance studies capture the details of the network architecture in sufficient detail but use abstract models of communication workload. The three communication attributes that are required for any network performance study are the message inter-arrival time distribution with the corresponding generation rate, the spatial distribution of messages or the traffic pattern and the mes-

sage size or volume. Typical synthetic environments have assumed that the temporal distribution is exponential, the spatial distribution is uniform or a few select localized communication patterns [16], and the message size is a fixed number of bytes. Such workloads may not necessarily mimic the real world behavior. Therefore, the performance results derived from synthetic workloads can provide a general guideline or bounding values, while it may be difficult to make cost-performance architectural design decisions using these results.

It is thus clear that any accurate performance evaluation of interconnection networks needs a marriage of the two viewpoints; evaluate the detailed network architecture using application characteristics. We use this new viewpoint in our study to get rid of the drawbacks of the earlier ones, yet combining their strengths.

Performance analyses of parallel systems with application benchmarks have been studied by a few researchers [22, 11]. But, none of these studies have investigated the network performance in particular. To our knowledge, the only reported work that has used parallel benchmarks to study network performance is by Kumar and Bhuyan [20]. They have used five shared memory benchmarks to analyze the impact of VCs and oblivious routing in a torus. It is shown that for the applications considered, VCs are helpful in minimizing the overall execution time and the depth of the buffers in each channel has a significant effect on network behavior. Also, their study complements the earlier claim that 4 VCs provide optimum performance. While their work provides the starting point of research in network performance evaluation with parallel applications, there are many open issues still unaddressed.

In this paper, we not only contradict their main results, but also look into many additional issues as summarized below.

- Our study investigates the performance benefits of VCs, routing adaptivity and flit buffers (which we refer to as network architectural enhancements throughout this paper) for five shared-memory applications. Unlike [20], this study is carried out accounting for factors which determine application scalability, such as the number of processor required and the application problem size.
- We also study the benefits of the above network architectural features under two different consistency models: sequential consistency (SC) and release consistency (RC).
- We investigate whether the performance benefits seen from the above are justified in terms of the cost of these features — that of slowdown in network cycle time — using a cost model presented in [2].
- We study how applications stand to benefit from increased router speeds due to simplistic router designs.
- Finally, we investigate the reasons for why increased network performance does not necessarily translate into benefits in application execution times and how some of the problems may be alleviated.

Using the communication characterization methodology presented in our earlier research [10], we simulate the execution of a suite of shared memory applications on our detailed network simulator implemented on SPASM [24]. The network simulator allows us to study the impact of VCs, and routing algorithms (deterministic and adaptive) on the network latency experienced by a message as well as on the overall execution time. The performance results show that there is a modest performance benefit with these enhancements in the average network latency. However, with respect to the overall execution time, this improvement is dwarfed in comparison to the other components which constitute the execution time. When considered in the context of application scalability in terms of the

number of processors and the problem considered, even though many of the considered applications inject a large number of messages into the network, their arrival into the network does not seem to generate any significant contention for network resources. Consequently, VCs and adaptive routing algorithms, which attempt to lower the network contention and not the raw network latency, do not show substantial saving in execution time. Thus, our results are in direct contrast with the results presented in [20], where applications with small and fixed problem sizes running on a fairly large and fixed number of processors were shown to benefit significantly from increased number of VCs and flit buffers.

Further, our results suggest that the performance rewards may not justify the cost of these enhancements unless an application is highly communication intensive and potentially scaling poorly. Our study also confirm the fears expressed in [2] that applications lose the moderate performance advantage of increased number of VCs if the cost for router slowdown due to VCs is considered. Rather, we feel that improving the raw network bandwidth by simpler and better router designs, exploring ways of reducing the number of network messages, tolerating the network latency, improved application mapping, better data partitioning and granularity control, will improve application performance.

The rest of this paper is organized as follows. Section 2 presents our performance evaluation approach, while the experimental platform is discussed in Section 3. The results from our evaluation studies and their discussion are presented in Section 4. Finally, Section 5 collates the contributions of this paper.

## 2 Performance Evaluation Approach

Our approach uses an execution-driven simulation system [24] for studying the network performance under application workloads. A schematic diagram of this system is shown in Fig. 1. There are two key steps involved in the simulation — *gathering communication events*, which involves parallel application execution until such point that processor threads need to communicate; and *simulating communication events* using the network simulator.

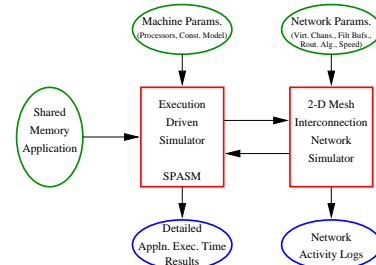


Figure 1: Simulation Environment

### Step 1: Gathering Communication Events

Execution of shared memory applications is closely intertwined with the communication capabilities of the underlying system. Hence, architectural changes in the network can have a significant impact on the application behavior. Thus, we resort to execution-driven simulation of the applications wherein there is a constant feedback between the communication event generator and the network simulator to ensure that events are generated and simulated in the correct order (as indicated by the two arrows between the network simulator and the execution-driven simulator in Fig. 1).

The simulation platform that has been chosen for this study is SPASM, a flexible parallel architecture simulator. SPASM can simulate a range of message-passing and shared memory platforms

[24]. As with several recent simulators [6, 14], SPASM does not simulate the details of the instruction execution. Instead, it only simulates instructions that may potentially involve network access such as LOADs/STOREs, and synchronization calls. The rest of the instructions execute at the speed of the native processor. The input to the simulator are applications written in C. The application code is preprocessed to label instructions that need to trap to the simulator, the compiled assembly code is augmented with cycle counting instructions, and the assembled binary is linked with the rest of the simulator modules. We assume a CC-NUMA architecture employing an invalidation-based cache coherence scheme using a full-map directory. SPASM also allows us to vary the memory consistency model of the shared memory platform.

## Step 2: Simulating Communication Events

To analyze the impact of applications on a mesh interconnection network, we feed the communication events generated in Step 1 to the network simulator. The network simulator models a bi-directional 2-D mesh. Each physical channel or link supports one or more virtual channels, each of which in turn has one or more flit buffers; connection between input and output links is established using a  $(5 \times 5)$  crossbar. The crossbar control is done based on the routing algorithm specified using a routing table implemented on the router. Events which involve communication invoke the network simulator with the necessary information (time-stamp, source, destination, and message size). The 2-D mesh simulator then performs a flit-level routing of the message using the above mentioned router model.

For the purposes of network simulation, we assume a synchronous network with fair arbitration between competing VCs (implying that all messages have equal priority). Note that out-of-order delivery of messages can occur between any given source-destination pair due to VCs and routing adaptivity. In-order delivery of such out-of-order messages is handled by the network interface simulated in SPASM. These in-order delivery guarantees are mandated by the shared-memory management layer in SPASM.

## 3 Experimental Setup

### 3.1 Architectural Platform

The parameters for the architectural details, used in the simulation with SPASM, are summarized in Table 1. The entire simulation is carried out in terms of CPU cycles. The write buffer size is set to 0 to model Sequential Consistency (SC) and to 4 for the Release Consistency (RC) simulations. Note that messages are of two sizes — those just containing a header (8 bytes) and those containing both the header and a cache block (40 bytes).

Cache Access	1 cycle
Memory Access	10 cycles
Message Header	8 bytes
Cache Block Size	32 bytes
Flit Size	8 bits

Table 1: Simulation Parameters

For each application, we mainly vary three network parameters. These are the number (1, 2, 4) of Virtual Channels (VCs), routing algorithm (oblivious, fully adaptive), number of flit buffers per virtual channel (1, 4, 8), and the network cycle time. Duato’s fully adaptive algorithm is simulated here [15]. Note that a minimum of 2 VCs are required to implement a fully adaptive routing algorithm

on a mesh network. Hence, the adaptive algorithm is simulated with 2 and 4 VCs.

### 3.2 Parallel Applications

A brief description of the applications used in this study is given below.

**1D-FFT:** 1D-FFT [24] implements a 1-dimensional complex Fast Fourier Transform. Each processor works on an assigned portion of the data space that is equally partitioned. There are three main phases in the execution. In the first and last phase, the processors perform the radix-2 Butterfly computation, which is an entirely local operation. The middle phase is the only phase involving communication in which the cyclic layout of data is changed to a blocked layout using an all-to-all communication step.

**IS:** IS is an *Integer Sort* kernel drawn from the NAS suite [3] that uses bucket sort to rank a list of integers. This application also has a regular communication pattern. The input data is equally partitioned among the processors. Each processor maintains local buckets for the chunk of the input list that is allocated to it. Hence, updates to the local buckets is an entirely local operation. There are two dominant phases in the execution that account for the bulk of the communication. In the first phase, a processor accesses a part of the local buckets of every other processor in making updates to the global buckets allotted to it. In the second phase, a processor locks each of the global buckets in ranking the list of elements allotted to it. The buckets are then merged using a logarithmic reduce operation and propagated back to the individual processors. Each processor then uses the merged buckets to calculate the rank of an element in its chunk of the input list.

**Cholesky:** Cholesky is an application drawn from the SPLASH benchmark suite [23]. This application performs a Cholesky factorization of a sparse positive definite matrix. The sparse nature of the matrix results in an algorithm with a data-dependent dynamic access pattern. Each processor, while working on a column, needs access to the non-zero elements in the same row position of the other columns. Once a non-local element is fetched, the processor can reuse it for the next column that it has to process. The allocation of columns to processors is done dynamically. Communication is involved in fetching all the required columns to the processor working on a given task. Hence, the communication patterns for Cholesky are not regular.

**Maxflow:** The Maxflow [1] application finds the maximum flow from a source to a sink, in a directed graph. Each processor accesses a local work queue for tasks to perform. These may in turn generate new tasks, which are added to this local work queue. Each task involves read and write accesses to shared data, which generate network traffic. The local queues of all processors interact via a global queue for load balancing. Most of the time is spent in data movement. The application exhibits dynamic data-dependent access patterns.

**NBody:** The NBody [23] application simulates over time the movement of bodies due to the gravitational forces exerted on one another, given some set of initial conditions. The parallel implementation statically allocates a set of bodies to each processor and goes through three phases for each simulated time step. The regions of space are organized in a tree data structure with the bodies forming the leaves. The first phase (make-tree) in each time step associates bodies to a specific region in space, and computes the center of mass for the regions hierarchically. In the second phase (consumer-phase), each processor computes the velocity component for the bodies assigned to it, which could involve visiting various nodes of the tree. In the third phase (producer-phase), each processor computes the coordinates of its bodies in space based on velocities computed in the previous phase. This application behavior is also dynamic in nature and may not exhibit regular communication patterns.

## 4 Performance Results

The performance metrics used in this study are the average network latency and execution time. Further, the execution time is broken down into four categories: CPU time, read stall time, write stall time and synchronization overhead (which may be composed of barrier stall time, mutex stall time and memory directory contention stall time). Even though we have experimental results for most applications across many these parameters, we present only the representative applications in this paper. Unless explicitly specified, results are presented for a network with 1 VC per physical channel, 1 flit-buffer per VC, a network cycle time identical to the processor cycle time and a sequential consistency model. Also, results for 8, 16 and 32 processors are for meshes of size  $(4 \times 2)$ ,  $(4 \times 4)$  and  $(8 \times 4)$  respectively. The oblivious routing algorithm (D) is simulated for 1, 2 and 4 VCs (denoted V1, V2 and V4, respectively) and the fully adaptive routing (A) is simulated for V2 and V4.

In the following discussion, the performance results from our experiments are organized into four parts followed by a discussion subsection. In the first part (section 4.1), we present performance results to study the impact of VCs and routing algorithms on application execution over a range of processors, application problem sizes and memory consistency models. In the second part (section 4.2), we examine the effect of adding flit buffers. In section 4.3, we discuss the cost associated with the addition of VCs in router design on the network cycle time. In section 4.4, we show that what we really need to do to improve application performance is to lower the network cycle time. Finally, section 4.5 discusses why the reduction in network latency for most network enhancements is not reflected in execution time.

### 4.1 Impact of Virtual Channels and Routing Algorithms

In a related study, Kumar and Bhuyan [20] have shown performance improvements for applications with the addition of VCs and flit buffers. But some of their performance graphs clearly indicate that a significant component of the overall execution time is spent in communication. In fact, in most cases, the communication times even dominate over the local computation performed by a processor. From our experience with this study, we feel that some of their results might be for poorly scaling applications. If executions of poorly scaling applications, are considered any marginal improvement in network latencies may have significant impact on execution times, and the importance of the network enhancements may be unduly exaggerated. For fear of making predictions with such situations, we have varied the number of processors and/or the problem size to make sure that the execution is scalable. Also, the memory consistency model of the shared memory system will have a bearing on the network traffic generated; thus the impact of the network enhancements on both SC and RC models has been studied.

### Varying the Number of Processors

Increasing the number of processors is likely to increase the network latency due to the larger network size. Further, the contention experienced by messages generated by each processor for network resources is also likely to increase. However, the local computation performed by a processor will also decrease. There is usually a point of diminishing returns beyond which if we increase the number of processors, the execution times can increase rather than decrease. Beyond this point, the execution is deemed unscalable and we do not wish to study the impact of network enhancements under such conditions. In the following experiment, we vary the number of processors for a given problem size and analyze the changes in the network latency and application execution time for different number of VCs and different routing algorithms.

Appln.	Proc	D,V1	D,V2	D,V4	A,V2	A,V4
FFT (32k)	8	31.88	29.99	29.99	31.63	31.38
	16	34.73	34.48	35.02	35.26	34.49
	32	50.37	49.77	49.90	47.83	47.05
IS (32k)	8	21.65	20.68	20.64	20.41	19.85
	16	25.09	24.39	24.32	23.44	22.15
	32	38.01	38.05	37.58	34.33	31.41
Cholesky (bcsstk14)	8	25.40	24.96	25.11	24.50	24.06
	16	30.78	29.69	29.77	28.25	27.82
	32	39.45	39.07	38.00	33.36	32.47
Maxflow (cube15.mod)	8	28.88	27.83	27.91	27.19	26.60
	16	37.31	34.03	33.29	-	30.91
NBody (infile)	8	28.96	28.33	27.80	27.39	26.77
	16	46.45	44.59	43.56	41.40	39.37

Table 2: Average Network Latency (in CPU cycles)

Table 2<sup>1</sup> shows the average network latency of the applications obtained from the detailed simulations. The results for Maxflow are shown only for 8 and 16 processors since the problem is observed to scale poorly. Results for NBody too are available only for 8 and 16 processors as the simulations for 32 processors take exceedingly large times to complete. We expect the average message latency to increase with the number of processors and the results confirm this. The network enhancements (VCs and routing strategies) are expected to show more improvements under higher network traffic conditions. The results indicate that the network latency decreases by minimal amounts with an increase in the number of VCs for oblivious routing. The fully adaptive scheme shows comparatively better performance than oblivious routing for all applications. The maximum benefit in delay is seen for IS and Maxflow which are seen to be scaling poorly (see Fig. 2). NBody and Cholesky show moderate improvement of about 15% in network latency between (D, V1) to (A, V4).

In Figure 2, we show the execution times of these applications with the different network enhancements. The performance results clearly show that the execution is scalable for all the cases except the execution of Maxflow on 16 processors and the execution of Integer Sort on 32 processors. FFT is the most scalable of the five applications, followed by NBody and Cholesky. As a very small fraction of the overall time is spent for communication in FFT, this application is the least likely to benefit from network enhancements. NBody and Cholesky get marginal benefits from these enhancements.

While the average network latency goes down by varying amounts with the enhancements, there is little performance gain in the overall execution time for all applications except that for the 16-node Maxflow. For instance, if we take the maximum processor configuration studied for an application, the performance improvement in execution times for FFT, NBody, IS and Cholesky are 0.5%, 1%, 5% and 7%, respectively as we move from (D, V1) to (A, V4). The maximum gain (26%) we find is in Maxflow with (D, V4), where we have specifically chosen a small (unscalable) problem to amplify the network effects. Reasons for this are the relatively large time spent in communication and the associated stall times. The adaptive algorithm does not perform as well as the deterministic routing strategy for this application because of latencies associated with in-order delivery for messages arriving out-of-order and the increased contention at the memory directory for the processor that handles the dynamic workload distribution.

In the case of NBody, the gains in network latencies due to VCs and adaptivity do not translate into equivalent gains in execution times for the application. Again, one of the reasons for this is

<sup>1</sup>Results for Maxflow with (A, V2) are unavailable.

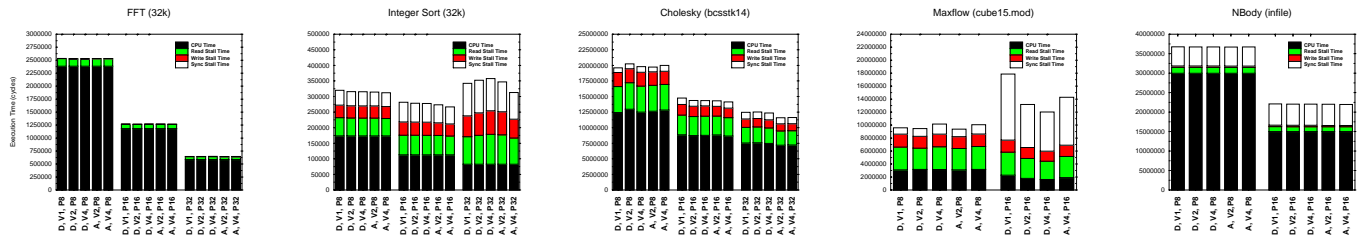


Figure 2: Impact of Processors (Execution Time in CPU cycles)

the additional latencies incurred outside the network for in-order delivery of messages. This and a few other anomalous increases in execution times seen across applications, despite decrease in corresponding network latencies, can be explained on the basis of the reasons presented in subsection 4.5.

### Varying the Problem Size

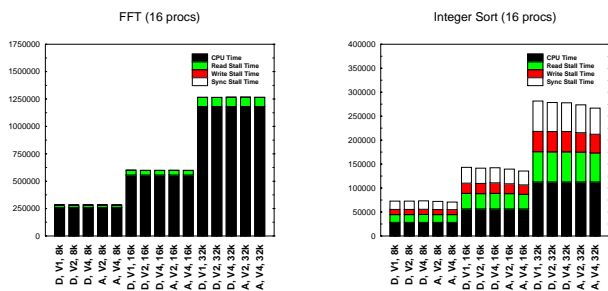


Figure 3: Impact of Problem Size (Execution Time in CPU cycles)

Simulation of Maxflow, Cholesky and NBody require an inordinate amount of time for large problem sizes. Therefore, we provide results only for 1D-FFT and IS with different problem sizes. Experiments were conducted for 8k, 16k and 32k problem sizes on a 16-node machine. Detailed execution time results are given in Figure 3, which shows the scaling of the application with problem size. Since communication becomes less important for larger problem sizes (in terms of the overall execution time), the VCs and the routing algorithm have a lesser impact for the larger problem sizes.

### Varying the Memory Consistency model

The two memory consistency models that we consider in this study are Sequential Consistency (SC) and Release Consistency (RC). These memory models define the programmer’s view of the memory state at different points in the execution. In SC, the CPU is stalled until the write is complete which may involve network actions to invalidate potentially cached copies. Only then is the CPU allowed to proceed. On a RC system, the CPU is not stalled. The write is simply registered in a write buffer, and the CPU is allowed to proceed. Only at a synchronization fence does the CPU stall to ensure that all pending writes in the write buffer are completed. The CPU may however need to stall if the write buffer is full.

Since there are fewer write stalls, we expect the CPU to pump data into the network at a faster rate on a RC system than on a corresponding SC system. As a result, the improvements with the network enhancements (VCs and adaptive routing) for RC are expected to be better than their SC counterparts. Figure 4 shows the performance results for the two memory models for IS and Cholesky. FFT has no writes and NBody has a very small write stall time; hence, these applications would not show much benefit under RC.

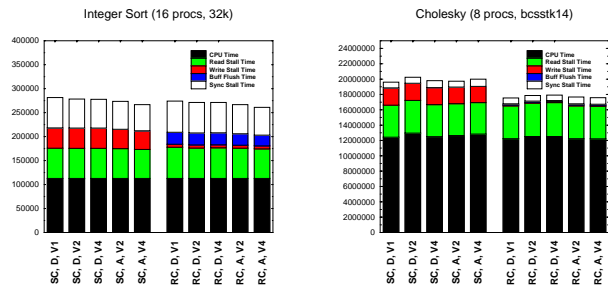


Figure 4: Impact of Memory Consistency Model (Execution Time in CPU cycles)

While the RC system shows better performance than the corresponding SC system because of hiding the write stall time, we do not see significant changes between the SC and RC improvements for VCs and adaptive routing. This suggests that the network contention even for RC does not warrant VCs and adaptive routing strategies.

Write buffer flushes cause bursty injection of messages into the network. However, because there is only one input and output port between the processor and the network router, a serialization of write messages occurs at this port resulting in significant buffer flush stall times when full or near full write buffers are flushed. Performance benefits are likely to result if multiple input/output ports between the processor and router are available. Of course, such additional ports are also likely to benefit other bursty arrivals to or departures from the network. These benefits would also be visible to applications under SC.

### 4.2 Impact of Varying Flit-Buffer Size

Deep flit buffers reduce “tail lengths” of messages, thereby reducing the effect of chained blocking. Thus, deeper flit buffers would show performance benefits when contention in the network is high.

Also, in practical networks, routers are clocked independently and are not synchronized with each other. In such networks, deep flit buffers help in transfer of flits between routers without the need for external links to be synchronized with or run at the same speed as the usually slower internal router clocks, thus helping overall network bandwidth to be high [8].

We present results for 1 to 8 flit buffers per VC, in Fig. 5. The results are reported for 16 processors with oblivious routing and 1 VC. The execution times do not show the benefit of deeper flit buffers, except for Maxflow. As mentioned earlier, Maxflow scales poorly for the problem considered. Also, the decrease in latencies (not shown here) with deeper flit buffers is negligible for the applications considered. Even though the tail length of the messages are reduced by deep flit buffers, the applications do not see the benefit because the effect of flit buffer size is visible only if the network has significant contention.

We also present results for a likely tradeoff — that of constant buffer space availability. Given this constraint, there are two ways

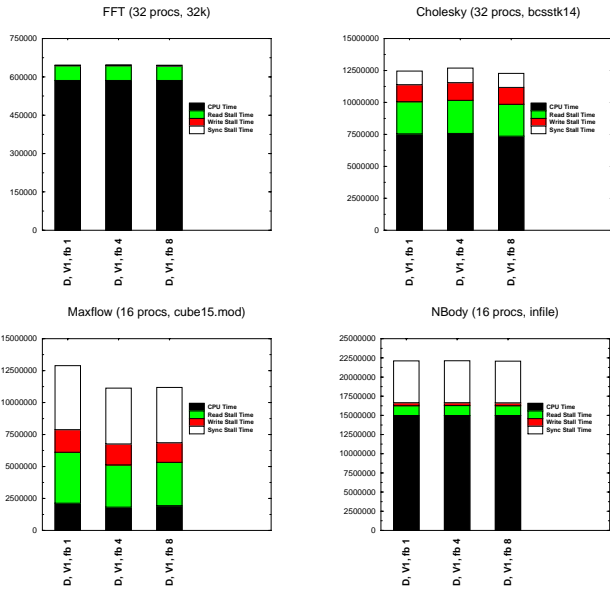


Figure 5: Impact of Varying Flit-Buffer Size (Execution Time in CPU cycles)

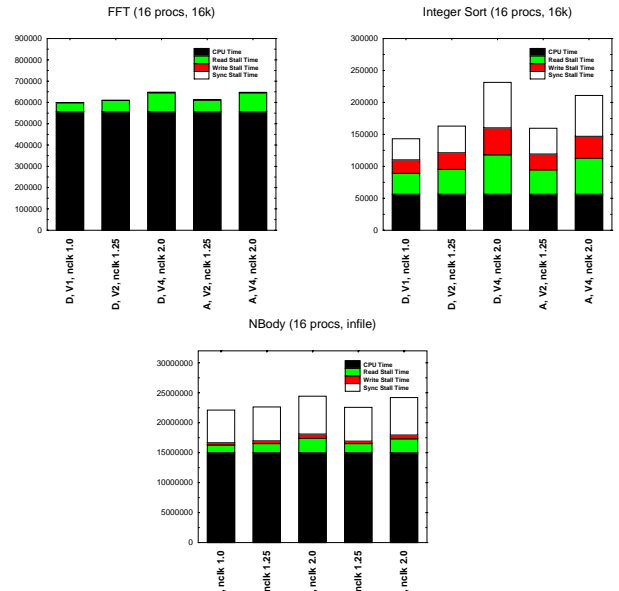


Figure 6: Impact of considering the cost of virtual channels (Execution Time in CPU cycles)

Appln.	Proc	Prob	D,V1,FB8	D,V2,FB4	D,V4,FB2
FFT	32	32k	49.68	50.11	50.69
IS	32	64k	36.72	37.44	38.79
Chol.	32	bcstk14	38.93	39.05	39.27

Table 3: Average Network Latency in CPU cycles (Deep Flit Buffers and Few VCs V/s Shallow Flit Buffers and More VCs)

to organize the available buffer space: either use deeper flit buffers with few VCs or use shallow flit buffers with more VCs. Results in Table 3 indicate that deeper flit buffers with few VCs is a better choice under this constraint.

### 4.3 The Cost of Virtual Channels

The discussions in the previous subsections implicitly assume that addition of VCs and adaptive routing do not increase the router clock cycle time. However, these architectural enhancements to network routers indeed increase their complexity. Current methods for implementing these features in routers indicate that this complexity also exacts a price — that of slowdown in router speeds. Aoyama and Chien [2] compare the cost of adding VCs and also that of various adaptive routing algorithms for wormhole switched routers. These costs come from the additional delay introduced into the critical path determining the router speed. These are due to the arbitration and multiplexing of competing VCs onto a physical channel (PC) and also due to crossbar controller speed (which is determined by the routing algorithm). Their delay models show about a 30% penalty in router speed per extra VC. For the purposes of this study, we assume (optimistically) a slightly smaller slowdown of 25% per extra VC<sup>2</sup>. Also, we do not associate any additional penalty for adaptivity. This is because modern wormhole routers use on-router lookup table based approaches for routing [21, 17, 8], which reduces the cost of crossbar control to a small constant independent

<sup>2</sup>In both cases, this slowdown is assuming a non-pipelined router model. It has been pointed out to us that a pipelined router is likely to suffer a lesser impact on the router clock, but may need additional pipe stages. We are currently investigating this in an independent study.

of the routing algorithm. However, these routers in majority, do not make dynamically adaptive routing decisions, though the tables can be reprogrammed to use alternate routing. Hence, the cost associated with making dynamically adaptive routing decisions is not clear.

A more realistic performance assessment of the benefits of VCs and adaptivity can be made using the above cost for each additional VC. Thus, network cycle time for 2 VCs per PC is inflated by 1.25 times that for a network with 1 VC per PC, and that for 4 VCs per PC is inflated two times. The results in Fig. 6 show that application performance consistently suffers with more VCs for both deterministic and adaptive routing. This indicates that the benefits of increased number of VCs and adaptivity in routing cannot compensate for the loss in performance due to slower speed of the router. These results thus tend to confirm the fears expressed by Aoyama and Chien in [2]. The cost of the VCs can be compensated only if the network latency due to additional VCs decreases at least by the corresponding amount (25–30 % per VC). However, for scalable applications, as used in this paper, this improvement is unlikely to occur.

The above results indicate that VCs may not justify the cost penalty. On the other hand, for bidirectional torus like architectures we need at least two VCs for deadlock-free routing. So use of VCs is inevitable for such topologies. The design philosophy in any case should be to use the minimal number of VCs.

### 4.4 Impact of Varying the Router Clock Cycle

One can observe from the results presented in the previous sections that network architectural enhancements such as increased VCs, adaptivity in routing and deep flit buffers can moderately increase application performance, if at all! Also whatever gains are observed are not consistent across all applications. So what is(are) the factor(s) that can most benefit application performance?

One such factor that benefits applications the most is the most obvious one — router clock cycle. In this experiment, we vary the network cycle time (2, 1, 0.5) with respect to the CPU cycle time and present results only for the (D, V1) case. Results presented in Fig. 7 only serve to confirm the obvious. It can also be seen that this factor benefits all applications consistently and in a very predictable

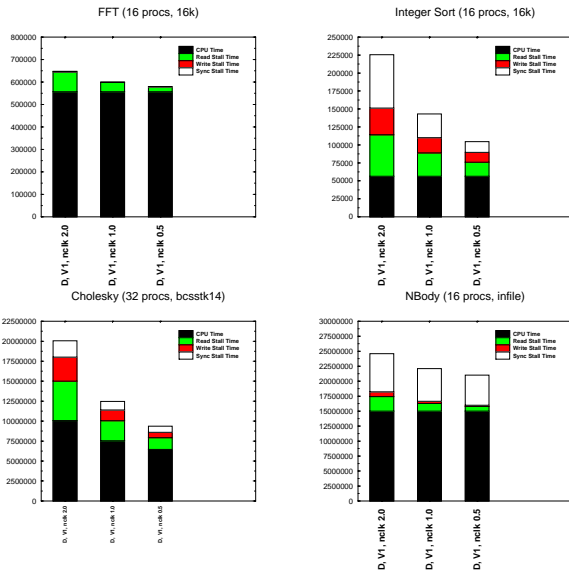


Figure 7: Impact of Router Clock Cycle (Execution Time in CPU cycles)

manner; and herein lies the strength of the obvious solution.

Improvements in router speeds can be gained through simplistic router design — first by using a minimal number of VCs. Additional improvements are likely to result from better router architectures and key issues to focus on include minimizing message header decode times, designing smaller and faster crossbars and their controllers, and by using wider data paths in the router.

#### 4.5 Lost Gains

It was seen earlier that the savings in network latency do not translate to proportional savings in execution time. One or more of the following reasons contribute to this.

**Asymmetric locality/network:** Gains in network performance are not uniform for all application threads. This can be attributed to asymmetric communication localities as well as the asymmetry of the mesh network.

**Barrier Synchronization:** Application requirements may dictate that threads synchronize after doing some computation. Application threads which benefited most (i.e. arrived at the barrier earliest) have to stall for the longest time waiting for the slowest thread (which benefited the least) to arrive at the barrier. Thus, the overall benefit seen by the application is that seen by the slowest application thread. We have observed this behavior in FFT and NBody executions.

**Dynamic Workload Allocation:** In applications where workload is dynamically allocated to processors, such as Cholesky and Maxflow which use the task-queue model, gains in network performance can be offset by workload imbalance, esp. for small problem sizes.

Application threads which benefit most from gains in network performance would most likely end up with a larger proportion of workload allocated to them. In some scenarios it is possible that these application threads execute the last few tasks needed to complete the application execution. The slight increases in application execution times observed in some cases for Cholesky and Maxflow despite decrease in average network latencies could be attributed to this reason (see Fig. 2 and Table 2).

**In order delivery requirement:** Using VCs and adaptivity can cause messages between a given source/destination pair of processors to be delivered out of order. The network interface has to ensure that this out of order delivery is not processor visible. This implies that messages which arrived out of order have to wait in network

interface buffers until their logical predecessors have arrived. This results in additional latency for such messages outside the network, thus offsetting at least some of the gains obtained by adaptivity and virtual channel flow control.

**Memory directory contention:** In applications where multiple remote messages, requiring memory directory access, arrive close together at a local processor, directory operations have to be serialized. This results in waiting messages suffering stall times at the memory directory resulting in performance losses. This effect has been observed in Cholesky, NBody and Maxflow. A possible solution to this problem is a multi-ported shared memory directory.

## 5 Concluding Remarks

This paper explores an area of interconnection networks which is not well studied — that of evaluating the performance of interconnection networks with real workloads. We have evaluated the benefits and the cost-performance tradeoffs in making architectural enhancements to a two dimensional mesh network for shared memory machines. This study was conducted using a set of five shared memory applications. Although limited in its scope due to network sizes considered, the number of applications used, number of architectural alternatives evaluated, and by practical considerations such as the cost of simulating larger problems; we believe that this study has been fairly conducted using a wide variety of of traffic and application scalability scenarios. The significant results of this paper are summarized below.

- We find that contrary to conventional wisdom, increasing the number of VCs and routing adaptivity offer little performance improvement for shared-memory applications. This observations holds true over a range of system and problem sizes and for both sequential and release consistency models, until applications start exhibiting poor scalability and inordinate amounts of time in communication.
- Addition of deeper flit-buffers also shows only modest performance benefits in network latency. However, deep flit buffers are necessary due to non-synchronous clocking among routers in practical interconnection networks implementations, as well as due to the difference in clock rates between external links and internal router logic in current routers [21, 8].
- Only a fraction of the modest gains in network latency due to VCs and adaptivity are passed on to the application performance, if at all! This is due to a combination of various causes mentioned in Section 4.5.
- When the performance improvement due to VCs is weighed against the decreased router speed due to addition of VCs, these gains are negated.
- The single most important factor for improving performance is the router speed. The benefits of a faster router are visible across all applications in a consistent and predictable fashion. Such benefits can be achieved by simpler and faster router designs, among other things with few or no VCs<sup>3</sup>, simple routing schemes, faster message-header decode times and small crossbar sizes.

Although our experiments are conducted for applications running on 16/32 node systems, we believe that our conclusions will apply for larger systems as well, when application scalability is taken into consideration. This is because the need for larger number of processors has to be justified with correspondingly larger problem

<sup>3</sup>Note that a few VCs may be necessitated by deadlock-free routing requirements in certain networks such as tori.

sizes and coarser communication granularities to keep the communication overhead bounded.

Our conclusions apply to shared-memory systems where all traffic has equal priority and message lengths are typically small and fixed. Also, we have conducted the study for uni-threaded and uni-programmed environments. These results may not apply to other environments, a few of which are discussed below.

- Application generated traffic patterns are likely to change in environments which support prefetching, multi-threading and multi-programming. Investigations on network performance need to address these issues as such environments become common in shared-memory systems.
- Message-passing machines and page-based distributed shared memory environments may exhibit different network performance as compared to the results in this paper, as message lengths in such systems are usually large (and in the former, variable as well).
- In environments where various classes of message traffic are supported with differing priorities or real-time constraints, VCs and prioritized arbitration among them may be necessary to meet system specifications. Our results are not applicable to such environments.

We have investigated benefits of adaptivity for a routing algorithm which requires additional VCs. Partially adaptive routing without additional VCs [18] might provide low cost benefits of adaptivity. Most importantly improving the performance of routers through architectural enhancements is likely to provide lasting payoffs.

## References

- [1] R. J. Anderson and J. C. Setubal. On the Parallel Implementation of Goldberg's Maximum Flow Algorithm. In *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 168–177, June 1992.
- [2] K. Aoyama and A. A. Chien. The Cost of Adaptivity and Virtual Lanes in a Wormhole Router. *Journal of VLSI Design*, 2(4):315–333, 1995.
- [3] D. Bailey et al. The NAS Parallel Benchmarks. *International Journal of Supercomputer Applications*, 5(3):63–73, 1991.
- [4] R. V. Boppana and S. Chalasani. A Comparison of Adaptive Wormhole Routing Algorithms. In *Proc. Intl. Symp. on Computer Architecture*, pages 351–360, May 1993.
- [5] Y. M. Boura and C. R. Das. Efficient Fully Adaptive Wormhole Routing in  $n$ -dimensional Meshes. In *Proc. Intl. Conf. on Distributed Computing Systems*, pages 589–596, June 1994.
- [6] E. A. Brewer, C. N. Dellarocas, A. Colbrook, and W. E. Weihl. PROTEUS : A high-performance parallel-architecture simulator. Technical Report MIT-LCS-TR-516, Massachusetts Institute of Technology, Cambridge, MA 02139, September 1991.
- [7] D. C. Burger and D. A. Wood. Accuracy vs. Performance in Parallel Simulation of Interconnection Networks. In *Proc. Intl. Symp. Parallel Processing*, April 1995.
- [8] J. Carbonaro and F. Verhoorn. Cavallino: The Teraflops Router and NIC. In *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, pages 157–160, August 1996.
- [9] A. A. Chien and J. H. Kim. Planar-Adaptive Routing: Low-cost Adaptive Networks for Multiprocessors. *Journal of the ACM*, pages 91–123, January 1995.
- [10] S. S. Chodnekar, V. Srinivasan, A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. Towards a Communication Characterization Methodology for Parallel Applications. In *Proc. Third Intl. Symp. High Performance Computer Architecture (HPCA-3)*, pages 310–319, February 1997.
- [11] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 2–13, May 1993.
- [12] W. J. Dally. Virtual-Channel Flow Control. *IEEE Trans. on Parallel and Distributed Systems*, 3(2):194–205, May 1992.
- [13] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. on Computers*, C-36(5):547–553, May 1987.
- [14] H. Davis, S. R. Goldschmidt, and J. L. Hennessy. Multiprocessor Simulation and Tracing Using Tango. In *Proceedings of the 1991 International Conference on Parallel Processing*, pages II 99–107, 1991.
- [15] J. Duato. A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks. *IEEE Trans. on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.
- [16] M. L. Fulgham and L. Snyder. Performance of Chaos and Oblivious Routers under Non-Uniform Traffic. Technical Report UW-CSE-93-06-01, Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195, July 1994.
- [17] M. Galles. Scalable Pipelined Interconnect for Distributed Endpoint Routing : The SGI SPIDER Chip. In *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, pages 141–146, August 1996.
- [18] C. J. Glass and L. M. Ni. A Turn Model for Adaptive Routing. In *Proc. Intl. Symp. on Computer Architecture*, pages 278–287, May 1992.
- [19] L. Gravano, G. D. Pifarre, P. E. Berman, and J. L. C. Sanz. Adaptive Deadlock and Livelock-Free Routing with All Minimal Paths in Torus Networks. *IEEE Trans. on Parallel and Distributed Systems*, 5(12):1233–1251, December 1994.
- [20] A. Kumar and L. N. Bhuyan. Evaluating Virtual Channels for Cache-Coherent Shared-Memory Multiprocessors. In *Proc. 10th ACM Intl. Conf. on Supercomputing*, May 1996.
- [21] S. L. Scott and G. M. 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, August 1996.
- [22] J. P. Singh, E. Rothberg, and A. Gupta. Modeling Communication in Parallel Algorithms: A Fruitful Interaction between Theory and Systems ? In *Proc. Sixth Annual ACM Symp. on Parallel Algorithms and Architectures*, 1994.
- [23] J. P. Singh, W.-D. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared-Memory. Technical Report CSL-TR-91-469, Computer Systems Laboratory, Stanford University, 1991.
- [24] A. Sivasubramaniam, A. Singla, U. Ramachandran, and H. Venkateswaran. An Approach to Scalability Study of Shared Memory Parallel Systems. In *Proceedings of the ACM SIGMETRICS 1994 Conference on Measurement and Modeling of Computer Systems*, pages 171–180, May 1994.