Improving the Performance of Bristled CC-NUMA Systems Using Virtual Channels and Adaptivity*

José F. Martínez, Josep Torrellas
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801 (USA)
{jmartinz,torrella}@cs.uiuc.edu

José Duato
Depto. Ingeniería de Sistemas y Computadores
Universidad Politécnica de Valencia
46071 Valencia (Spain)
jduato@gap.upv.es

Abstract

Current high-end parallel systems achieve low-latency, high-bandwidth network communication through the use of aggressive design techniques and expensive mechanical and electrical parts. High-speed interconnection networks, which are crucial to achieve acceptable system performance, may account for an important fraction of the total cost of the machine. To reduce the network cost and still maintain scalability, bristled configurations, in which each router connects to several processing nodes, pose an attractive alternative. Their lower bandwidth, however, may adversely affect the efficiency of the parallel codes.

In this paper, we show how virtual channels and adaptive routing can make bristled systems more attractive: overall performance improves in congested scenarios while remaining practically unaltered under light traffic conditions. Experimental results are obtained by using execution-driven simulation of a complete state-of-the-art CC-NUMA system, with dynamic superscalar processors and contemporary pipelined routers. The results show that, in bristled hypercubes with 2 processing nodes per router, SPLASH-2 applications with significant communication run 5-15% faster if we make use of virtual channels and adaptive routing. The resulting systems are only 1-10% slower than systems with non-bristled hypercubes and similar routing support, even though the former only need about half of the network hardware components present in the latter. Additionally, virtual channels and adaptivity are shown to be of negligible effect in non-bristled hypercubes.

1 Introduction

Hardware-coherent distributed shared-memory architectures are a promising way of building scalable, easy-to-program multiprocessor systems. Distributed organisations rely on an interconnection network to communicate and to synchronise between processing nodes, or PNs. Such a network can be basically described as a set of ASIC routers, and a set of wires establishing the interconnection, which we call links. Routers carry the responsibility of correctly routing and delivering messages from any source to any destination, whereas links must ensure sufficient speed and bandwidth, as well as reliable signal transmission.

While interconnection networks attempt to deliver messages with low latency and high bandwidth, there are constraints in VLSI technology, chip area, and chip pin count that limit the network's capabilities. For example, chip pin count restrictions may limit the number of ports that a router can have. This limitation may, in turn, limit the number of dimensions that a given network topology may have. A case in point is the SGI Origin2000 hypercube network, which is built out of six-port routers [7]. Of these six ports, four are used to link to other routers, and the remaining two host PNs. This hardware allows for hypercube topologies of up to 16 routers; for larger systems, a less optimal fat-hypercube approach is utilised [11].

Thanks to continuous advances in high-density ASIC fabrication, high pin-count packaging and clock speeds, it is possible to increase the number of ports per router and, therefore, build larger, lower-latency networks. It is evident, however, that such larger networks come at the expense of a higher system cost, caused by the greater number of routers and wires, as well as by the added complexity. This cost increase can be critical to tight-budget customers.

A solution that allows for a higher number of PNs without expanding the interconnection network is called *bristling*. Bristling consists of connecting several PNs to each router; each PN can contain one or more processors. A network with p PNs connected to each router is said to have a *bristling factor p*. A direct network with bristling factor p comprising n network nodes or routers will be referred to as a *p-way n-node network*. For example, if we want to use a six-port router as the building block for a 16-PN system, we can furnish it with either a 4-way 4-node network or, with one port per router left unused, a 2-way 8-node or a 1-way 16-node network (see Figure 1).

An obvious major advantage of bristled networks is that, for a given number of PNs, they need only a fraction of the hardware present in their corresponding non-bristled configuration. Consequently, if cost is at a premium, bristled networks can be a useful choice. It is necessary, however, that the router be *symmetric*, which means that all its ports are identical. An asymmetric router, which offers a dedi-

*This work was supported in part by the National Science Foundation under grants NSF Young Investigator Award MIP-9457436, ASC-9612099 and MIP-9619351, DARPA Contract DABT63-95-C-0097, NASA Contract NAG-1-613, NCSA Grant MIP980001N, the Bank of Spain, and gifts from IBM and Intel.

To appear in the 1999 ACM International Conference on Supercomputing.

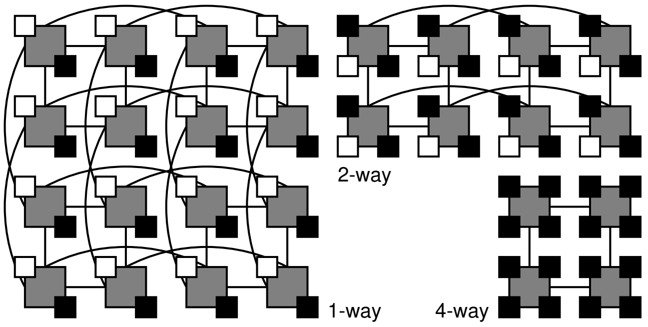

Figure 1: 1-,2-, and 4-way bristled networks, each allocating a 16-PN system, using a 6-port router. Black squares indicate PNs; white squares denote unused ports.

cated interface for PNs at one or more ports, is less flexible. For example, while the SGI Origin2000's SPIDER router is symmetric [7], the Cray T3E's router is not [14].

An additional advantage of bristled networks is that they have a lower average node distance, which may reduce the average latency. In the example above, the 1-way 16-node hypercube has an average distance of 2 hops, whereas the 2-way 8-node and the 4-way 4-node bristled networks have 1.5 and 1 hops, respectively. It can be easily derived that, in an n-dimensional hypercube, the average node distance is $n/2$, while the total number of links is $n \cdot 2^{n-1}$. As we decrease n, the average node distance goes down arithmetically, whereas the total bandwidth does so geometrically. Consequently, under light traffic, a bristled hypercube could have an edge over its non-bristled counterpart. However, if the network is heavily used, the lower bandwidth of the bristled network may become a bottleneck. As a result, techniques to diminish the impact of the lower bandwidth would be highly desirable to make bristled systems more attractive.

Two such techniques are *virtual channels* and *adaptive routing*. Virtual channels (VCs) allow messages to share a common physical link [2]. They have been typically used to improve physical link utilisation by allowing messages to overtake blocked ones on a shared physical link, and to avoid deadlocked situations. For example, some topologies like tori, usually require two links between every pair of nodes to ensure deadlock freedom in most routing algorithms; these two links may be implemented over a single physical connection by using VCs. Similarly, systems with cache-coherent, non-uniform memory accesses (CC-NUMA) usually require disjoint networks for requests and replies to avoid protocol deadlock. VCs may again be used for this purpose.

Most existing routers perform *deterministic routing*, which selects one, and only one outgoing path for each incoming message in a router, regardless of traffic conditions [3]. Adaptive routing, on the other hand, takes into consideration traffic conditions as seen locally by the router, and applies a selection function to choose among several output path options to make the message advance. Adaptive routing often needs multiple paths between neighbouring routers to guarantee absence of deadlock; these can once again be provided through VCs. It is very common for the adaptive routing function to contain a *escape subfunction*, which is usually a deterministic routing function guaranteeing a *escape route* to every message if the adaptive options are not available [5].

VCs and adaptive routing have been extensively studied in the past [2, 6, 8, 13]. Most of the research has been based on simulations using synthetic workloads, or at most traces of real applications. In addition, adaptive routing has seen a few actual implementations, like in the Cray T3E network [14]. Many of the evaluations using synthetic workloads concluded that both VCs and adaptivity were beneficial. It was frequently possible to push the network to the limit by increasing the message injection rate and by using very long messages. On the other hand, CC-NUMA traffic is mainly characterised by message sizes limited to roughly that of a cache line, and it follows largely unknown patterns, at times bursty. In addition, past trace-driven simulations could not make use of any feedback from the traffic and contention conditions to the issue rates of the processors. All these simplifications are likely to influence the results.

There have been only two studies that have evaluated the impact of VCs and/or adaptive routing on the performance of CC-NUMA systems through execution-driven simulation. The first one, by Kumar and Bhuyan [10], concluded that the use of VCs on a torus was a winning choice, whereas the other one, by Vaidya *et al* [18], pointed out a lack of scalability in the experiments conducted in the former work, and claimed no advantage in using extra VCs. The authors also concluded that adaptivity was not beneficial. We comment on these works in Section 5.

Overall, it is a common belief that VCs and adaptivity have little impact on the performance of CC-NUMA systems. Today's low-latency, high-bandwidth networks are hard to put under pressure in most common situations. Nevertheless, as we have pointed out before, the fact that bristled networks trade off cheaper configurations for a serious reduction in bandwidth portrays a scenario in which the above may not hold. Consequently, in this work we revisit the concepts of VCs and adaptivity in bristled hypercubes.

Our results show that, in bristled hypercubes with 2 PNs per router, SPLASH-2 applications with significant communication run 5-15% faster if we make use of virtual channels and adaptive routing. The resulting systems are only 1-10% slower than systems with non-bristled hypercubes and similar routing support, even though the former only need about half of the network hardware components present in the latter. Additionally, virtual channels and adaptivity are shown to be of negligible effect in non-bristled hypercubes.

This work is organised as follows: Section 2 presents the network model and routing algorithms; Section 3 describes the simulation environment utilised to obtain our results; Section 4 presents the results; Section 5 discusses related work; finally, Section 6 concludes.

2 Network Model

2.1 Router Design

Our model is a pipelined router that follows today's design and technology trends. Figure 2 depicts a block diagram of a 2-port version. It represents an aggressive base design that leaves little margin for further performance improvements.

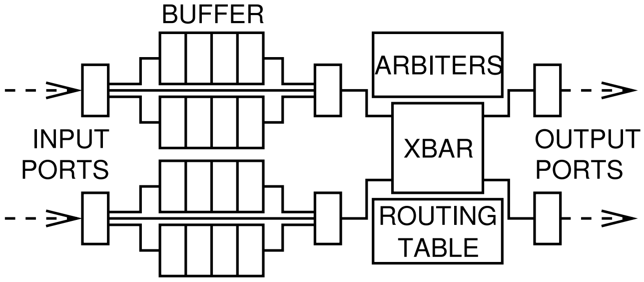

Figure 2: Block diagram of a two-port instance of the router model utilised in this work. Note the input and output latches, the VCs (two per port in this case), and the bypass routes to the crossbar.

Information is exchanged in the form of messages, which can be cache lines, requests, or coherence primitives like invalidations and acknowledgments. Each message is composed of a number of *flow control digits*, or *flits*, which represent the amount of information in a message that a router can process each cycle. Such flits are in turn composed of *phits*, which are the transmission unit on the physical links connecting the routers.

Each router chip is furnished with a number of bidirectional, clock-decoupled, wave-pipelined ports [15], operating at four times the frequency of the chip and able to accept a new phit every clock cycle independently of the wire length. Each phit is 16-bit wide and each router flit is composed of four phits, thus matching link and router bandwidth while keeping the number of pins per port relatively low. Each port connects to a de-serialising input latch that builds one flit out of every four phits, and from there to as many FIFO input buffers as VCs, as well as directly to the crossbar through a single bypass route. Each buffer can store over two of the largest possible messages in the system. Adaptive routes cannot be selected unless there is enough space at the other side of the link to store the message completely, so flow control in adaptive VCs is effectively virtual cut-through. This is in compliance with Duato's premises for deadlock freedom as described in [4]. Output ports are serialising latches that decompose each flit back into four phits.

The input and output latches smoothen the transition between links and router, which have different width and clock rate. A 1-cycle arbitration and decoding process determines if the flit can pass through the crossbar to its output port or if, instead, it must be stored in the corresponding buffer. The former constitutes a *bypass*, which can only occur when the flit is a data flit following a winner, or when it is a header flit and there are no contenders, much in the same way as in the SPIDER router chip [7]. If such conditions are met, the flit will cross during the next clock cycle to its output port; otherwise, it will be stored in the corresponding buffer. Bypass arbitration and buffer selection are performed in parallel. This is possible even for the header, due to the utilisation of look-ahead routing, as explained below. Overall, it takes a minimum of three router clock cycles to perform a network hop. If routers have a clock skew, an extra cycle should be included to synchronise the data with the internal clock. We have not included the effect of clock skew in our simulations.

A slower 2-cycle central arbiter is responsible for resolving the conflicts among the candidates at the head of the buffers and fetching the winners. Messages carry no priority information, hence a round-robin scheme is performed during arbitrations. Up to one buffer per port can be granted permission at any given cycle. The arbiter tries to preserve as much continuity of transmission as possible: once a message acquires an output port, it will flow without interruption until it runs out of space at the next stage or it bubbles. Enforcing continuity usually results in reduced average latency. When a message sees its flow interrupted, its input and output ports are released and become available to other messages.

Whenever a header flit crosses toward its output port, a table look-up is performed in parallel to determine its output(s) at the next router; this scheme, called *look-ahead routing*, saves one cycle per hop with respect to conventional inplace routing. If adaptive routing is implemented, all routes are retrieved from the table and annotated in the header of the message. This may pose time and space problems in systems with fully adaptive algorithms and a high number of dimensions, since route selection at the port of entry may become too slow, and header and tables may lack enough space to allocate all the options. Restricting the number of routes in fact gets rid of this problem. In our case, we only consider routing through at most two physical links.

The whole design resembles a hybrid of the SGI SPIDER and T3E chips. We have mimicked SPIDER's general pipelining scheme, arbitration procedure, and look-ahead routing. On the other hand, we have kept T3E's FIFO buffers and adaptive routing scheme.

2.2 Routing Algorithms

Five minimal routing algorithms have been used in this study. We have encoded their behaviour into names as described below. The basic features that distinguish them are: number of VCs, adaptivity, and VC sharing.

1×VC (baseline): Deterministic scheme with one VC for cache coherence protocol requests and another for replies. The only port supplied by the routing function is the one the *e-cube* function would choose [3]. Such a port corresponds to the lowest dimension in which the current and the destination routers differ, or to the PN if we are at the end of the journey. We will call this port the *deterministic* port. This base function will constitute the escape subfunction for all the others [5].

2×VC: Scheme like 1×VC plus one extra VC for requests and another one for replies, for a total of two VCs per message type. Messages consider the extra VC before the escape route.

Sh2×VC: Scheme like 2×VC, except that the two extra VCs are shared among requests and replies. The selection order of the extra VCs for requests is the complement of the one for replies to balance request and reply traffic. For example: if a request message must consider extra VC α first and extra VC β next, then a reply message ought to consider β before α. Thus, the total number of possible paths for a message is three.

Ad2×VC: Adaptive extension of 2×VC that adds to the set of possible paths the two extra VCs at the port advancing

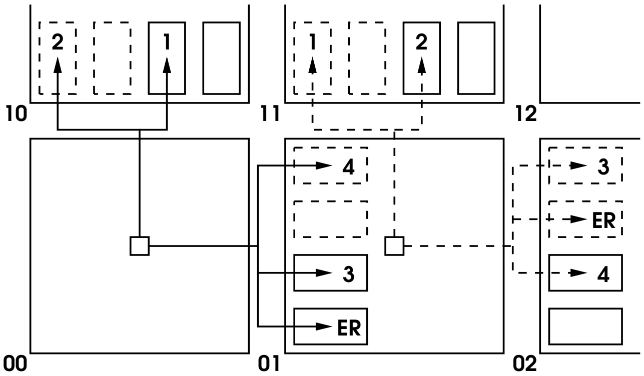

Figure 3: Routing example with ShAd2×VC for request and reply messages advancing toward router 12. Selection order is indicated by numerals. The message and the VCs belonging to the reply subnetwork are dashed. Escape routes are labelled as ER.

in the highest dimension yet to be traversed, for a total of up to three paths per message. We call the port at such a dimension the *adaptive* port. If only one dimension needs to be crossed, this port will coincide with the deterministic port. In all cases, the paths through the adaptive port will be considered first, followed by the VCs at the deterministic port.

ShAd2×VC: Combination of Sh2×VC and Ad2×VC. The output ports considered are the ones in Ad2×VC. However, now the extra VCs at both ports are shared between request and reply traffic, as in Sh2×VC, making five the possible paths per message. The selection order for requests and replies is also complementary at the adaptive port. Figure 3 depicts typical VC selections for individual request and reply messages under this algorithm. Notice how all the routing algorithms described above are subfunctions of this one.

3 Simulation Environment

We have developed a fully parametric network simulator that models the router and routing algorithms described above. We have integrated this tool in a detailed simulator of a CC-NUMA system with dynamic superscalar processors. We perform simulations under a system based on MINT [19] that has been modified to support accurate and efficient execution-driven simulations of dynamic superscalar processors [9].

The simulated CC-NUMA system has 32 single-processor PNs. Each processor is modelled as a 4-issue dynamic superscalar with three pipelined integer units, two pipelined add/multiply floating point units, one non-pipelined division unit and two load/store units. The processor has a 128-entry instruction window and a 2048-entry, 2-bit branch prediction buffer. It can support up to 16 and 8 outstanding loads and stores, respectively. The processor clock rate is 1GHz. We use the release memory consistency model. Routers cycle at 250MHz, but link decoupling allows links to run at 1GHz. The networks simulated are hypercubes built out of 7-port routers. We examine the following configurations: 4-way 8-node, 2-way 16-node, and 1-way 32-node hypercubes. We have assumed that all topologies are implemented in three

Traffic T (Bytes/Op.)	*SPLASH Application*
$.40 \leq T < 1$	**FFT**, **Radix**
$.20 \leq T < .40$	**Ocean**, Cholesky
$.10 \leq T < .20$	**LU**, Water-Nsq, Water-Sp
$0 \leq T < .10$	Barnes, FMM, Raytrace, Radiosity, Volrend

Table 1: Classification of SPLASH-2 applications according to the level of communication bandwidth required. The codes chosen in our experiments are typed in bold face.

Element	*Delay (Cycles)*	*Configuration*
L1 Cache	2	8 Kbytes
L2 Cache	25	32 Kbytes
Local Memory	150	4-Kbyte Pages
Network	12-20 per Hop	Hypercube

Table 2: Basic system characteristics used in the experiments. Delays are expressed in processor cycles. Cache and memory delays correspond to a round trip access from the processor.

physical dimensions. We derived wire lengths and delays using the formulas in [15]. However, for the system sizes that we are analysing, the experimental values are not much different from what we obtained using a uniform single-unit link delay. Therefore, we are presenting results under the latter assumption. An invalidation-based cache coherence protocol similar to the one in DASH [12] is used. The protocol is able to correctly handle out-of-order messaging.

We choose four parallel shared-memory codes from the SPLASH-2 suite [20]: FFT, Radix, Ocean, and LU. The choice is made to be representative of the different levels of communication bandwidth required in the suite. Table 1 summarises the levels of communication bandwidth required by the SPLASH-2 applications for 32 processors as explained in [20], with the ones chosen in bold face. We pick two applications from the class with the highest bandwidth requirements, and then one from each class for which bristling may cause a significant performance loss. Applications with bandwidth requirements at or under the level of Water show virtually no change in the execution times, and hence we do not include them in this study. Nevertheless, we did run Water with all our configurations and verified that none of the optimisations evaluated in this paper had any effect.

To make sure that the experiments are scalable and the results are meaningful, we conduct our simulations for the cache parameters recommended in the SPLASH-2 paper [20]. Specifically, first- and second-level cache sizes are chosen so that the applications can fit their smaller working set but not the larger one. Caches have 64-byte lines and are direct-mapped. Table 2 lists the basic system parameters that we use in all the simulations. All cache and memory latency values correspond to typical round-trip requests from the processor in the absence of contention. As a reference figure, a round trip from a processor to the memory in another PN in an unbristled 32-node system takes an average of 310 processor cycles.

4 Experimental Results

We run simulations for all the applications in CC-NUMA machines with 4-way 8-node, 2-way 16-node, and 1-way 32-node hypercubes, all of them with 32 processors. A 1-way 16-node hypercube is also used in some experiments. We simulate all five routing algorithms described in Section 2. We examine the execution time and then the traffic in the network. The reported execution times include all aspects of the architecture, not only the network.

4.1 Execution Time and Speedup

Figure 4 compares the execution times under different network configurations and routing algorithms, all of them normalised to that of the 1-way 32-node system with 1×VC. Note the different scaling of the plots. It can be observed that all the applications increase their execution time when a bristled configuration is adopted. When using the baseline routing algorithm (1×VC), Radix takes 30% more time to execute on a 2-way network and almost 120% more on a 4-way network. Less dramatically, Ocean and FFT's execution times grow approximately by 15% and 55% on a 2-way and a 4-way network, respectively. LU, on the other hand, only suffers a 25% increase with a 4-way network. These results show that, as bristling reduces total bandwidth in the network, applications find it harder to push their messages through it. As we increase the bristling factor, the small reduction in the average node distance is overwhelmed by the effect of lower bandwidth. The same general trend is observed in the bars indicating execution times for the other routing algorithms.

The figure also shows that, in 1-way network configurations, neither extra VCs nor adaptivity are able to reduce the execution time much. At most, they reduce it by 5%. This is in agreement with results obtained by Vaidya *et al* [18], although it contradicts the results of the same authors in [17] for pipelined designs. When network ASICs and wires are fast enough, it is seldom the case that the network clogs long enough for extra VCs or the adaptive algorithm to noticeably improve the performance.

However, if we look at the behaviour of bristled networks, one can see how both extra VCs and adaptive routing help in holding back the execution time growth. For example, in Radix, VCs alone (2×VC and Sh2×VC) reduce execution time by 8% and 20% in 2- and 4-way networks, respectively. Moreover, such percentages further improve to 15% and 28% respectively if we add adaptivity. FFT and Ocean also see improvements that range from 5 to 15% for the combined effect of VCs and adaptivity in the bristled networks. Although LU sees nearly no change with extra VCs alone, it does show a 5% improvement over the baseline algorithm when using adaptivity in the 2-way and 4-way systems.

Sharing of VCs by request and reply messages does not bring additional benefits. The differences between 2×VC and Sh2×VC, and between Ad2×VC and ShAd2×VC, are usually negligible. This holds for all the applications considered.

Another way of looking at the impact of bristling, VCs, and adaptivity is to depart from a 1-way 16-node system and see what speedup applications reach when the number of processors is doubled. We can double the number of PNs by

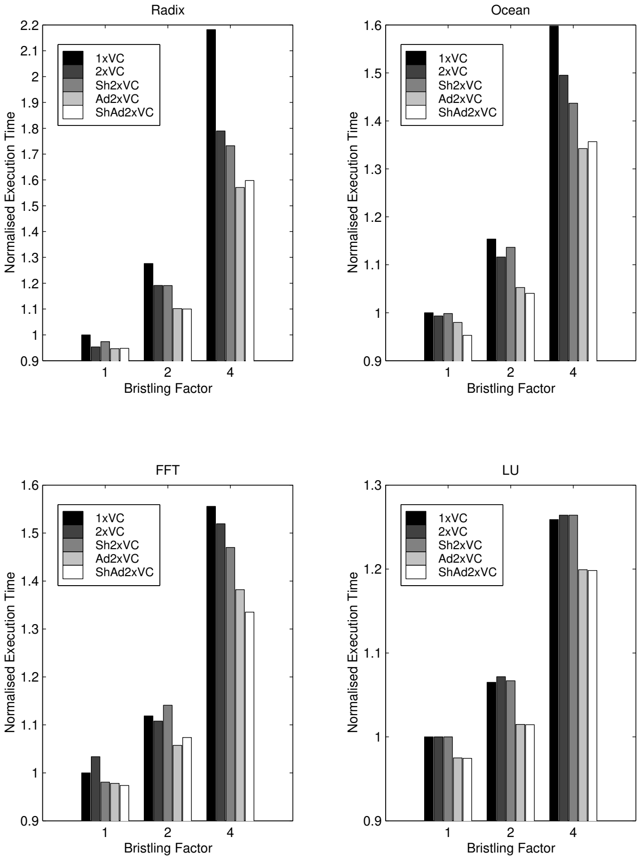

Figure 4: Normalised execution time versus bristling factor.

upgrading to an unbristled 1-way 32-node system or, instead, upgrading to a bristled 2-way 16-node system. Although upgrading to a 4-way 8-node system makes no practical sense, we have included the latter to have another view of the performance degradation. Figure 5 shows the outcome of the experiment. It is organized as Figure 4.

Figure 5 shows that, under 1×VC, bristled systems attain much lower speedups than their unbristled counterparts. Again, mind the different scaling of the plots. Radix, for example, achieves a 1.75 speedup in the non-bristled system, but only 1.4 in the 2-way system. Interestingly, however, the version with VCs and adaptivity (Ad2×VC) is able to deliver a 1.6 speedup on the 2-way system. This figure is quite close to the 1.75 speedup achieved in the 1-way configuration, even though the total bandwidth and the number of components in the network are about half. This trend repeats in all other applications. Overall, the 2-way systems are only 1-10% slower than the 1-way systems with 1×VC.

In the case of 4-way bristled systems, although the extra VCs and adaptive routing help a bit, the speedups are very limited. Radix even experiences a slowdown when using 1×VC. This points out that, despite the higher link utilisation that extra VCs and adaptivity enable, total raw bandwidth is of chief importance.

Overall, the data suggests that, among hypercube networks, configurations with a bristling degree of 2 that use extra VCs and adaptivity are very cost-effective. Indeed, extra VCs and adaptivity decrease the execution time of

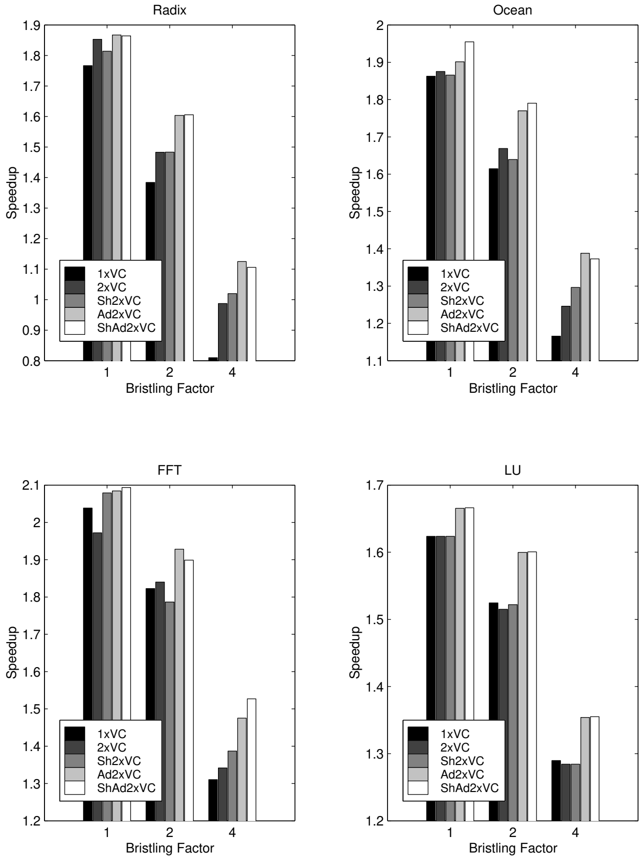

Figure 5: Speedups of different 32-PN systems over the 1-way 16-node 1×VC system.

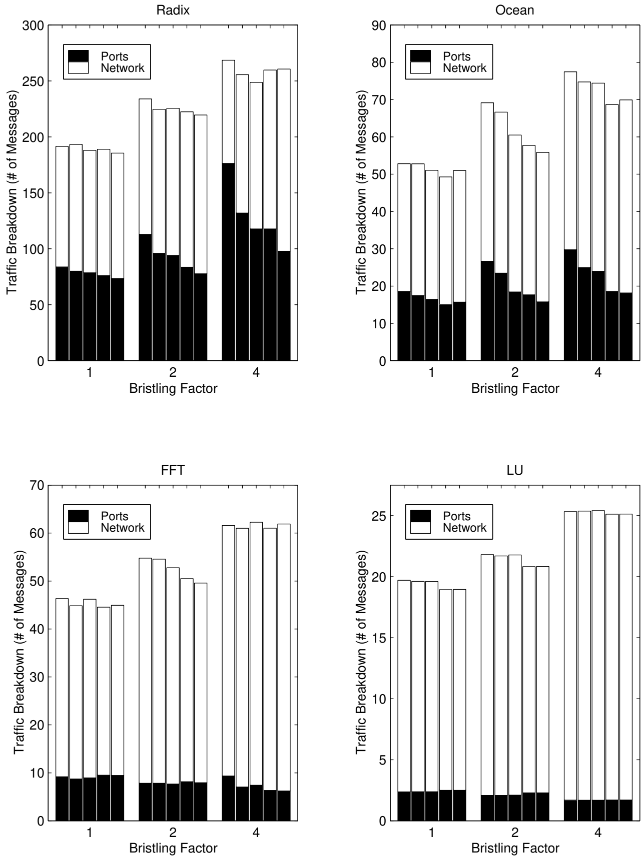

Figure 6: Average number of messages in transit anywhere in the network in a given cycle. For a given bristling factor, the bars correspond to the different routing algorithms as in the previous figures.

high-communication applications in 2-way bristled networks by 5-15%. As a result, 2-way bristled networks are only 1-10% slower than non-bristled networks with baseline routing. This is in spite of the fact that the former use about half of the network hardware of the latter. A second observation is that the sharing of the extra VCs among requests and replies does not lead to noticeably better results. Finally, the speedup of the non-bristled configurations shows that all applications were scalable at the points considered.

4.2 Traffic and Contention Break-Down

To understand better the impact of extra VCs and adaptive routing in bristled configurations, Figures 6 and 7 show the average traffic (number of messages in transit anywhere in the network at any given cycle) and contention (number of cycles wasted to contention in the network in the round trip of a message) respectively. The data are broken down into the contribution at the PN injection ports and in the network. The wider choice of routes that extra VCs and adaptive routing offer to message headers should help loaded input ports inject more messages per time unit, and should also allow messages to achieve a higher mobility in a busy network.

Figure 6 shows that, in many cases, the traffic in a given bristled network stays largely constant as the routing algorithm varies. However, as shown in Figure 7, the cycles wasted to contention in a given bristled network decrease significantly as we use more advanced routing algorithms that include extra VCs and adaptivity. Constant traffic with lower contention simply implies a higher network throughput and, as we have shown in Figure 4, a faster execution time.

The decrease in contention is especially high in the application with more traffic, namely Radix. Note the scale of the figure for Radix. The extra VCs and the use of adaptivity make more routes available for each message, resulting in faster message transfers. In the case of LU, the traffic is already fairly low. Consequently, reductions in contention losses as shown in Figure 7 do not reflect in a significantly lower execution time (Figure 4).

Figures 6 and 7 show that, in applications with moderate traffic, most of the in-transit messages and contention cycles are in the network body, and not at the injection ports. However, the latter become the bottleneck as traffic becomes higher. For example, in Radix, the contribution of the injection ports to the in-transit messages and contention cycles is about 50%.

As a result, the decrease in contention cycles as we add extra VCs and adaptivity shown in Figure 7 is largely due to different reasons in different applications. In traffic-intensive applications like Radix and, to a lesser extent, Ocean, the decrease is due to the contribution at the injection ports. This means that messages that used to wait in the ports

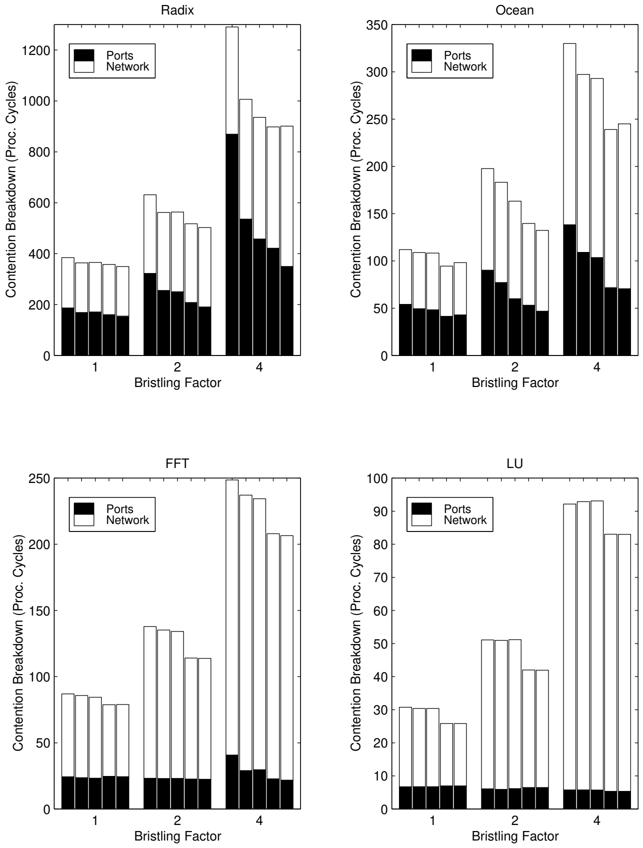

Figure 7: Average number of processor cycles wasted to contention in the network in the round trip of a message. For a given bristling factor, the bars correspond to the different routing algorithms as in the previous figures.

are now slithering through the network. The added features increase the *injection capacity* of the ports.

However, in the applications with less traffic like FFT and LU, contention decreases after messages have been injected. This suggests that messages do not encounter difficulty in getting injected into the network. The benefits of the extra VCs and the adaptivity come from augmenting the *mobility* of the messages in the network.

5 Related Work

Although the study of the performance of interconnection networks in multiprocessors is a popular topic, our work has focused on the specific issue of interconnection networks for CC-NUMA systems. It is widely accepted that, to capture many of the patterns of the traffic produced by real applications, execution-driven simulations are required. Under such environments, the contention caused by traffic in the network affects the issue rate of the processors, which in turn affects the pressure on the network. Most of the work done in the past, however, has been done using synthetic workloads, or at most trace-driven simulation. Even the few studies that have used execution-driven simulation and real applications have generally based their conclusions on relatively outdated router and system models.

Aoyama and Chien [1] proposed a parametric delay model of a router chip. Such a model takes into account the complexity introduced by the number of ports, virtual channels, and adaptive routing. Although it was used by researchers to time their designs, it is not a suitable model for pipelined routers and links, which enable a shorter clock cycle by decoupling the different components of the chip. In the design of the high-performance router utilised in our study, we have instead used a contemporary pipelined model with reasonable timing values.

The effect of extra VCs and adaptivity in CC-NUMA systems has been studied by several authors. Kumar and Bhuyan [10] utilised execution-driven simulations to determine the impact of the number and depth of VCs on the performance of parallel applications running on a CC-NUMA system. Using a torus as the topology, they found that both factors reduced the execution time. A later work by Vaidya *et al* [18] further explored the influence of network parameters on the execution time of parallel applications. Unlike the former study, this second work found the use of VCs and adaptivity to have a negligible effect, sometimes even adverse. Vaidya *et al* attributed the disagreement to Kumar and Bhuyan's use of non-scalable application configurations.

Unfortunately, none of these two studies used pipelined routing chips or decoupled links. In particular, the results of Vaidya *et al* rely on Aoyama and Chien's delay model to claim negative influence of VCs in the system performance. As Vaidya *et al* pointed out in their paper, these conclusions cannot be extrapolated to a pipelined router, which is the common case in today's state-of-the-art systems.

Finally, Vaidya *et al* later published another paper [17] in which they explored the effect that some of the techniques utilised in today's pipelined routers have on message latency. They drove their network model with synthetic patterns and gradually introduced improvements such as look-ahead routing, reduced routing tables, adaptive routing, and path-selection heuristics. Although their study showed noticeable impact on message latency and addressed many implementation details, it did not show the effect on actual codes. In our study, we see that the effect of extra VCs and adaptivity on real codes is actually negligible in full hypercubes. It is only noticeable in bristled networks, where the bandwidth is limited.

6 Conclusions

This work has shown that virtual channels (VCs) and adaptive routing can help improve the performance of bristled CC-NUMA systems and make them quite attractive to the budget-minded designer. Specifically, among hypercube networks, configurations with a bristling degree of 2 that use extra VCs and adaptivity are very cost-effective. Indeed, 2 extra VCs and adaptivity decrease the execution time of high-communication applications in 2-way bristled networks by 5-15%. As a result, 2-way bristled networks are only 1-10% slower than non-bristled networks with baseline routing. This is despite the fact that the former use only about half of the network hardware of the latter. Furthermore, light-traffic applications are not adversely affected by the use of extra VCs or adaptive routing. Finally, extra VCs and adaptive routing are of very limited advantage to non-bristled systems.

We have confirmed that the major contributions of extra VCs and adaptive routing to the improved execution times of the applications are greater port injection capacity and message mobility in the network.

In the future, it will be interesting to combine adaptivity and DAMQ buffers. DAMQ buffers [16] are a feature present in the SPIDER chip that prevents messages from blocking others residing in the same VC but going to different output ports. The combination of both features may further improve the overall performance of a CC-NUMA. Note that combining adaptivity and DAMQ buffers can be done by simply adopting a storage policy for the incoming messages, for example to store the message in the input buffer corresponding to its escape route.

Acknowledgements

We thank Venkata Krishnan, Liuxi Yang, Anthony Nguyen, Marcelo Cintra, Steve Scott, and the referees for their feedback. José Martínez is supported in part by a fellowship from the Bank of Spain. Josep Torrellas is supported in part by an NSF Young Investigator Award.

References

[1] K. Aoyama and A. A. Chien. "The Cost of Adaptivity and Virtual Lanes in a Wormhole Router". *Journal of VLSI Design*, Vol. 2, No. 4, 1995.

[2] W. J. Dally. "Virtual-Channel Flow Control". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 3, No. 2, March 1992.

[3] W. J. Dally and C. Seitz. "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks". *IEEE Trans. on Computers*, Vol. C-36, No. 5, May 1987.

[4] J. Duato. "A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 7, No. 8, August 1996.

[5] J. Duato. "A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 6, No. 10, October 1995.

[6] J. Duato. "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No. 12, December 1993.

[7] M. Galles. "Scalable Pipelined Interconnect for Distributed Endpoint Routing: the SGI SPIDER Chip". *Proc. Symp. High Performance Interconnects (Hot Interconnects 4)*, August 1996.

[8] P. Gaughan and S. Yalamanchili. "Adaptive Routing Protocols for Hypercube Interconnection Networks". *IEEE Computer*, May 1993.

[9] V. Krishnan and J. Torrellas. "A Direct-Execution Framework for Fast and Accurate Simulation of Superscalar Processors ". *Intl. Conf. on Parallel Architectures and Compilation Techniques*, October 1998.

[10] A. Kumar and L. N. Bhuyan. "Evaluating Virtual Channels for Cache-Coherent Shared-Memory Multiprocessors". *Proc. 10th ACM Intl. Conf. on Supercomputing*, May 1996.

[11] J. Laudon and D. E. Lenoski. "The SGI Origin: A CC-NUMA Highly Scalable Server". *Proc. 24th Intl. Symp. on Computer Architecture*, June 1997.

[12] D. E. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. "The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor". *Proc. 17th Intl. Symp. on Computer Architecture*, May 1990.

[13] L. M. Ni and P. K. McKinley. "A Survey of Wormhole Routing Techniques in Direct Networks". *IEEE Computer*, Vol. 26, No. 2, February 1993.

[14] S. L. Scott and G. M. Thorson. "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus". *Proc. Symp. High-Performance Interconnects (Hot Interconnects 4)*, August 1996.

[15] S. L. Scott. "The Impact of Pipelined Channels on k-ary n-cube Networks". *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 1, January 1994.

[16] Y. Tamir and G. L. Frazier. "Dynamically-Allocated Multi-Queue Buffers for VLSI Communication Switches". *IEEE Trans. on Computers*, Vol. 41, No. 6, June 1992.

[17] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. "LAPSES: A Recipe for High-Performance Adaptive Router Design". *Proc. Intl. Symp. High-Performance Computer Architecture*, January 1999.

[18] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das. "Performance Benefits of Virtual Channels and Adaptive Routing: An Application-Driven Study". *Proc. 11th ACM Intl. Conf. on Supercomputing*, July 1997.

[19] J. E. Veenstra and R. J. Fowler. "MINT: A Front-End for Efficient Simulation of Shared-Memory Multiprocessors". *Proc. 2nd Intl. Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, January 1994.

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". *Proc. 22nd Intl. Symp. on Computer Architecture*, June 1995.