Performance Evaluation of Networks of Workstations with Hardware Shared Memory Model Using Execution-Driven Simulation*

J. Flich, M. P. Malumbres, P. López and J. Duato
Dpto. Informática de Sistemas y Computadores
Universidad Politécnica de Valencia
Camino de Vera, 14, 46071–Valencia, Spain
E-mail: {jflich,mperez,plopez,jduato}@gap.upv.es

Abstract

Networks of workstations (NOWs) are becoming increasingly popular as a cost-effective alternative to parallel computers. Typically, these networks connect processors using irregular topologies, providing the wiring flexibility, scalability, and incremental expansion capability required in this environment. Similar to the evolution of parallel computers, NOWs are also evolving from distributed memory to shared memory programming model. However, physical distances between processors are longer in NOWs than in tightly-coupled distributed shared-memory multiprocessors (DSMs), leading to higher message latency and lower network bandwidth. Therefore, the network may be a bottleneck when executing some parallel applications in a NOW supporting a shared-memory programming paradigm.

In this paper we analyze whether the interconnection network is able to efficiently handle the traffic generated in a NOW with the shared memory model. In particular, we are interested in analyzing the influence of the routing mechanism in the performance of the system. We evaluate the behavior of a NOW with irregular topology by means of an execution-driven simulator using SPLASH-2 applications as the input load. The results show that the routing algorithm can considerably reduce the total execution time of applications. In particular, routing adaptivity can reduce the total execution time by 58 % in some applications. These results confirm the behavior observed in previous works using synthetic traffic loads.

1 Introduction

Research in parallel computers has focused on multicomputers and multiprocessors during the last decades. These machines provide high computing power by arranging tens or hundreds of processors that work in a coordinated way in order to solve a given task. The communication needed to coordinate and synchronize these processors is carried out through the interconnection network.

Large-scale parallel computers evolved from multicomputers (iPSC/1, iPSC/2, nCUBE-2, nCUBE-3, TMC CM-5, Intel Paragon) to distributed shared-memory multiprocessors (DSM), either with cache coherence (SGI Origin 2000) or without cache coherence (Cray T3D, Cray T3E). The higher architectural complexity required to provide a single memory address space is worth its cost. It provides a simpler programming model in which communication is achieved by accessing shared memory locations, and synchronization is performed by using barriers. Additionally, sequential applications that require a large amount of memory to be executed efficiently can be directly ported to DSMs. In general, this is not possible in a multicomputer because each processor has a relatively small local memory.

However, due to the increasing computation power of microprocessors and the high cost of parallel computers, networks of workstations (NOWs) are currently being considered as a cost-effective alternative for small-scale parallel computing. Although NOWs do not provide the computing capacity available in multicomputers and multiprocessors, they meet the needs of a great variety of parallel computing problems at a lower cost. Moreover, the nature of NOWs allows an incremental expansion of the system.

Nevertheless, most commercial interconnects for NOWs only provide support for message-passing. These networks consist of a network interface card that is plugged into the I/O bus of each workstation, and one or more switches interconnecting the interface cards through point-to-point links [3]. In general, the bandwidth provided by currently available interface cards and switches is high enough for the requirements of message-passing applications. However, the performance of some parallel applications is limited by

*This work was supported by the Spanish CICYT under Grant TIC97–0897–C04–01 and by Generalitat Valenciana under Grant GV98-15-50

message latency.

Similarly to the evolution of parallel computers, NOWs are also evolving from distributed memory to shared memory. Most approaches implement the shared memory model in software [1, 28]. However, a few commercial interface cards support the shared-memory programming model. This is the case for the SCI-PCI adapter from Dolphin [6]. This card does not support cache coherence in hardware because memory traffic cannot be seen from the I/O bus. Also, interface cards are reaching the limits of I/O buses.[1] In order to provide a higher bandwidth and lower latency, some researchers have started to develop interface cards that are plugged directly into the memory bus providing support for shared-memory [18], also implementing a cache-coherence protocol.

2 Motivation

A decade ago, the invention of wormhole switching led to very fast interconnection networks [2]. As a consequence, the interconnection network was no longer the bottleneck in a multicomputer. However, DSMs require faster interconnection networks than multicomputers because messages are shorter (a typical message consists of a cache line and some control information) and much more frequent [15]. While the interconnection network is not the bottleneck yet, some researchers reported that a lower latency and a higher network bandwidth may significantly reduce the execution time of several parallel applications [29, 4, 27]. As processor clock frequency is increasing at a faster rate than network bandwidth, the interconnection network may become a bottleneck within the next years [8].

The situation is more critical for NOWs supporting a single memory address space. Physical distance between processors is higher in NOWs than in DSMs, leading to higher latency (due to the propagation delay) and lower bandwidth (due to the use of narrower links).[2] Additionally, the use of irregular topologies makes routing and deadlock handling much more complex. Existing routing algorithms are either deterministic [3] or provide some degree of adaptivity [20, 19, 26]. Some of them lead to a frequent use of non-minimal paths and an unbalanced use of physical links. As a consequence channel utilization is low, reducing the effective network bandwidth even more. Therefore, we can expect the network to be a bottleneck when executing some parallel applications on a NOW supporting a shared-memory programming paradigm. Then, it is important to analyze the behavior of parallel applications on these machines.

In this paper, we analyze whether the interconnection network in a NOW is able to handle the traffic generated by the execution of some parallel applications. In particular, we evaluate different routing algorithms on NOWs with irregular topology, using the total execution time as the main performance parameter. These results will provide a more precise idea of the impact of routing algorithms on application performance than previous results based on trace-driven simulations [27]. As applications, we have selected a subset of the SPLASH-2 suite [30]: FFT, Radix, Barnes, and LU. The execution of these applications is simulated on a DSM test-bed with a hardware supported cache-coherence protocol. This environment allows the execution of a parallel program on the simulated machine, using a given cache-coherence protocol that generates coherence commands and memory traffic to a detailed interconnection network simulator subsystem. The main goal of this study is to determine whether the routing schemes affect the total execution time of an application, in order to evaluate their impact on the network design. We show that the use of adaptive routing helps to significantly reduce the execution time of these applications.

The rest of this paper is organized as follows. In Section 3, NOWs are introduced, describing several techniques to improve performance. Section 4 presents the test-bed used in the evaluation. In Section 5, the performance of different routing algorithms is evaluated using SPLASH-2 applications. Finally, in Section 6 some conclusions are drawn.

3 Networks of Workstations

NOWs are usually arranged as switch-based networks with an irregular interconnection pattern. Some of the switch ports are attached to processing elements, while the rest of the ports are connected to other switches to provide connectivity between processors, or left open. Generally, links connecting switches are bidirectional full-duplex.

Most of the NOWs proposed up to now implement wormhole switching [3, 14]. Since this is the most commonly used switching technique, we will restrict ourselves to wormhole switching in this paper.

Routing in irregular networks is more complex than in regular ones, due to the irregularity and unpredictability of the network topology at design time. Two approaches can be used: source routing and distributed routing. In source routing, when a processor generates a message it looks up in a routing table that provides the path towards the destination processor and stores that information in the message header. Thus, the path followed by a given message is fixed at the source node. This approach is used in Myrinet networks [3]. On the other hand, in distributed routing each switch has a

[1] For example, link bandwidth in Myrinet is 160 Mbytes/s. ServerNet II [11] and ChaosLAN [17] provide links with 1 Gigabit/s peak bandwidth. However, a 32-bit PCI bus running at 33 MHz only achieves a peak bandwidth of 133 Mbytes/s.

[2] For example, Cray T3E, Cray T3D, and SGI SPIDER routers [23, 22, 10] use 14, 16, and 20 data wires per link, respectively. ServerNet II [11] and Myrinet [3] use serial and 8-bit links, respectively.

routing table. When a message has to be routed at some switch, the routing algorithm looks up in the associated table, obtaining the output link to be used. Therefore each switch computes only the next link to be used, leading to more routing flexibility. In both cases, before the network is ready to deliver messages, some network configuration algorithm must be executed in order to fill the routing tables with the suitable information. This information depends on the routing algorithm. Several distributed deadlock-free routing schemes have been proposed for irregular networks, like the up*/down* routing scheme [20], the Eulerian-trail routing algorithm [19] or the adaptive routing scheme proposed in [26, 24].

We will restrict the scope of this paper to distributed routing, analyzing the behavior of NOWs with up*/down* and adaptive routing. To make the paper self-contained, we will summarize these routing schemes in the following sections.

3.1 Up*/Down* Routing

Up*/down* is a distributed deadlock-free routing algorithm that provides partially adaptive routing in irregular networks. In order to fill the routing tables, a breadth-first spanning tree (BFS) on the graph of the network is computed first using a distributed algorithm. Routing is based on a direction assignment to all the links in the network. In particular, the "up" end of each link is defined as: (1) the end whose switch is closer to the root in the spanning tree; (2) the end whose switch has the lower ID, if both ends are at switches at the same tree level. The result of this assignment is that each cycle in the network has at least one link in the "up" direction and one link in the "down" direction. To avoid deadlocks, this routing scheme uses the following up*/down* rule: a message cannot traverse a link in the "up" direction after having traversed a link in the "down" direction.

When a message arrives at a switch, the routing algorithm is computed by accessing the routing table. The address of the table entry is obtained by concatenating the input port number with the address of the destination node stored in the message header. If there are several suitable outgoing ports, one of them is selected.

Up*/down* routing is not always able to provide a minimal path between every pair of nodes due to the restriction imposed by the up*/down* rule. As the network size increases, this effect becomes more important.

3.2 Adaptive Routing

We will summarize the design methodology for adaptive routing algorithms on irregular networks proposed in [24, 26]. This methodology starts from a deadlock-free routing algorithm, splitting all the physical channels in the network into two virtual channels. We will refer to them as the *original* and *new* channels, respectively. Next, the routing algorithm is extended so that new channels are freely used with the only restriction that they must forward messages closer to their destination, and original channels are used in the same way as in the original routing function. Additionally, when a message is injected into the network, it can only leave the source switch through new channels, since they provide a higher degree of adaptivity and, usually, a shorter path. Also, when a message arrives at an intermediate switch, it first tries to reserve a new channel. If all the suitable outgoing new channels are busy, then an original channel belonging to a minimal path is selected. If none of the original channels provides a minimal path to the destination, then one of the original channels that provide the shortest path will be used. To ensure that the new routing function is deadlock-free, once a message reserves an original channel, it can no longer reserve a new one [24, 26]. This message will be routed through original channels until it arrives at the destination switch.

The minimal adaptive algorithm [24, 26] is an application of this design methodology to the up*/down* routing algorithm. This routing algorithm provides fully adaptive minimal routing between all pairs of nodes until messages are forced to move to original channels. When a message starts using original channels, it provides the same adaptivity as the up*/down* routing algorithm.

4 EDINET: An Execution-Driven Simulator to Evaluate Interconnection Networks

The EDINET (Execution Driven Interconnection NETwork) simulator [9] allows executing a shared-memory application on a simulated DSM or NOW system. It is composed of two simulators. The first one is Limes [16], an execution-driven simulator that allows parallel program execution and models the memory subsystem. The second one is the interconnection network simulator that we have already used in several evaluation studies [24, 7]. The memory simulator part of Limes simulates the memory subsystem sending requests to the interconnection network simulator in order to simulate the transmission of messages.

4.1 Processor and Memory Model

The processor model has been chosen based on modern processor designs [13]. We assumed at each node a 480 MHz single-issue microprocessor with a perfect instruction cache and a 128 KB 2-way set associative data cache with a line size of 64 bytes. The memory bus was assumed to be 8 bytes wide. On a memory block access, the first word of the block was assumed to be returned in 16 processor cycles;

the successive words follow in a pipelined fashion, one per clock cycle. The machine used a full-mapped directory with an invalidation-based cache coherence protocol [5] implemented in hardware. The network interface has two separated queues to process incoming and outgoing requests. To avoid coherence protocol deadlocks we have implemented the approach proposed in [5]. In this approach when an input buffer is full, requests are rejected with a NACK command. Later, the source will send again the request after a random delay.

We have used a sequential memory consistency model. In this model, there is at most one outstanding request per processor. Therefore, a data miss stalls the processor until the data is returned. So, there is a bounded number of messages traversing the network at the same time. We have used this model because it is the simplest to implement and the easiest to program. Other memory consistency models may help in improving performance by increasing processor utilization [12]. So, with more flexible consistency models more traffic may be generated, and thus, processors may issue more requests to the network.

4.2 Network Model

The network is composed of a set of switches. Network topology is completely irregular and was generated randomly, taking into account three restrictions. First, we assumed that there are exactly 4 nodes (processors) connected to each switch. Second, all the switches in the network have the same size. We assumed that each switch has 8 ports (so, there are 4 ports available to connect to other switches). Finally, two neighboring switches are connected by a single link.

Each switch has a routing control unit that selects the output channel for a message as a function of its destination node, the input port number, and the output channel status. Table look-up routing is used. Up*/down* and adaptive routing (see section 3) may be used. Routing delay is assumed to be three processor clock cycles[3]. The routing control unit can only process one message header at a time. It is assigned to waiting messages in a demand-slotted round-robin fashion. When a message gets the routing control unit, but it cannot be routed because all the alternative output channels are busy, it must wait in the input buffer until its next turn. A crossbar inside the switch allows multiple messages traversing it simultaneously without interference. It is configured by the routing control unit each time a successful routing is made. The time needed to transfer a flit across the crossbar is assumed to be 3 processor clock cycles. Considering that wires in NOWs are usually long, and assuming a Myrinet link bandwidth of 160 MB/s and maximum wire length of 10m, physical channel propagation delay is assumed to be equal to 36 processor clock cycles. Transmission of data across channels is pipelined [21], assuming that a new flit can be injected into the physical channel every 3 processor clock cycles. Flits are one byte wide. Physical links are one flit wide.

As physical channels may be split into several virtual channels, the "stop and go" flow control protocol [3] is used to efficiently use the available link bandwidth. In this protocol, the receiving switch transmits a stop (go) control flit when its input buffer fills over (empties below) 66% (40%) of its capacity. Considering that there can be up to 12 flits on the wire and assuming a one-cycle delay to decode control flits, input buffer size must be equal to 75 flits. Output buffer size has been fixed to 4 flits.

[3] We assume that the router clock runs at 160MHz, as in current Myrinet switches. This clock frequency is three times slower than processor clock frequency. Routing takes one router clock cycle, that is, three processor clock cycles. In what follows, all the references to clock cycles will refer to processor clock cycles.

Table 1. Applications and the input sizes used.

Application	Problem Sizes
LU	16×16 to 512×512 doubles, 8×8 blocks
Radix	256K, 512K, and 1M keys, 1K radix, max 1M
FFT	2^{12} to 2^{18} complex data points
Barnes	1024, 2048, and 4096 particles

4.3 Network Load

We injected into the network the traffic generated by the execution of several parallel benchmarks. In particular, we used a subset of the SPLASH-2 suite (FFT, LU, Radix, and Barnes) [30]. These are challenging computational applications. Table 1 lists the actual problem sizes used for the applications.

5 Performance Evaluation

In this section, we evaluate the impact of using different routing algorithms on total execution time and network behavior, using the execution-driven test-bed introduced above. In particular, the up*/down* (UD) and minimal adaptive (MA-2VC) routing algorithms are compared. As MA-2VC algorithm requires two virtual channels per physical channel, in order to make a fair comparison, we have also evaluated the up*/down* routing algorithm with two virtual channels (UD-2VC). Also, we have included an ideal network model called PNET (Perfect NETwork). In this model, messages cross the interconnection network without contention using minimal paths. Message latency is

assumed to be equal to the base latency, that is, the propagation delay along the shortest path from source to destination.

We have evaluated a system with 64 nodes. Taking into account that there are 4 processors connected to each switch, the network has 16 switches.

The main performance measure is the total execution time of the applications. Also, other measures like average message latency and average network throughput have been considered. In particular, we obtained the average message latency and average throughput during program execution at regular intervals. In all simulations the applications finished correctly and the application results were the expected ones.

5.1 Simulation Results

Figure 1 shows the total execution time of the applications for the routing algorithms analyzed.

In the FFT kernel (Figure 1-a), the MA-2VC routing algorithm reduces the total execution time by 38 % for the lowest problem size and by 53 % for the highest problem size with respect to the UD. When comparing with UD-2VC, MA-2VC reduces total execution time by 25 % for the lowest problem size and by 36 % for the highest problem size. These results show that the key to improve performance is not only adding virtual channels but using them in a more flexible way, as the MA-2VC does. This adaptive routing algorithm allows messages to follow alternative paths instead of blocking, waiting for channels to become free. Additionally, it balances link utilization, improving throughput (as we will see later) and providing a higher effective network bandwidth.

On the other hand, PNET reduces execution time from 42 % for the lowest problem size to 43 % for the highest problem size with respect to the MA-2VC. Although results obtained by MA-2VC are still far from the ones obtained by the PNET model, the relative difference remains constant for all the complexity points evaluated. Comparing with UD, PNET outperforms UD by 64 % for the lowest problem size and by 73 % for the highest problem size.

In the LU, Radix, and Barnes applications (Figures 1-b, 1-c, and 1-d) results are qualitatively similar. The MA-2VC outperforms both the UD and UD-2VC in terms of application execution time. The largest improvements are achieved for the highest problem size and are equal to 38 % (LU), 58 % (Radix), and 46 % (Barnes) when comparing MA-2VC with UD, and equal to 19 % (LU), 40 % (Radix), and 30 % (Barnes) when comparing MA-2VC with UD-2VC. As can be seen, adaptivity and virtual channels are very useful in all the cases.

Adaptive routing helps in reducing application execution time. However, it is important to characterize the network traffic requirements of the applications. Network throughput and latency indicate how loaded the network is.

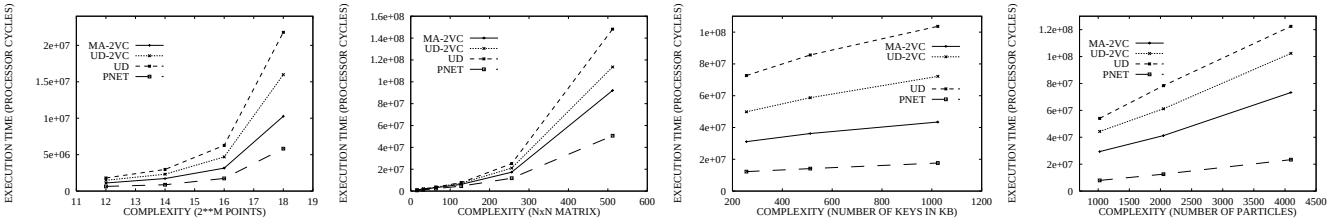
Figure 2 shows the average latency measured as the elapsed time from message generation at the source node to message delivery at the destination node for the applications considered. Again, as expected, average message latency is reduced when using MA-2VC routing algorithm.

Figure 3 shows the average network throughput (measured in flits/cycle/switch) achieved by each application for every routing algorithm analyzed. As can be expected, network throughput increases when application complexity increases. The exceptions are Radix and Barnes, in which network throughput is almost constant for all the sizes of the data set we analyzed. In all the cases, the use of MA-2VC routing strongly increases network throughput with respect to UD routing. The explanation is very simple. The amount of generated messages is almost the same, regardless of the routing algorithm. As we have seen, MA-2VC reduces total execution time by allowing the use of shorter and alternative paths to forward messages toward their destinations. As a consequence, the ratio between the number of flits transferred and the elapsed time increases.

On the other hand, maximum throughput never exceeds 0.2 flits/cycle/switch. In [26], it was shown that network throughput could be as high as 0.08, 0.15, and 0.25 flits/cycle/switch using UD, UD-2VC, and MA-2VC routing algorithms, respectively, for a uniform distribution of message destinations and a similar network topology. As a consequence, the network is not saturated, on average. But there may exist hidden hot spots that saturate the network.

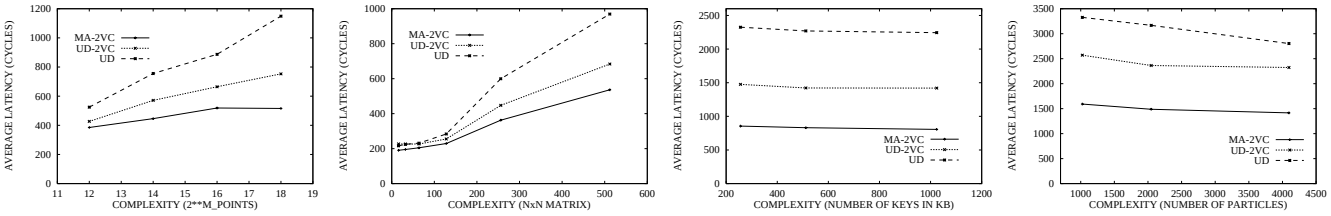
The analysis of traffic rate and average latency during the execution of the application allows us to analyze application behavior more precisely. Moreover, we are interested in analyzing how good adaptivity is in the presence of hot spots. We have gathered statistics for 1000 points at regular intervals during all the simulation. For the sake of brevity, results are only shown for the FFT and Radix applications and for the largest data set analyzed (2^{18} points in FFT and $1M$ keys in Radix). Figures 4 and 5 show the average traffic rate and average latency during the sampling intervals, for both applications considered.

As can be seen, in the FFT application the three curves for throughput and latency have almost the same shape, with three phases of high traffic rate separated by synchronization points, in which traffic rate is very low. The main differences are quantitative. The MA-2VC routing algorithm achieves the highest throughput and the lowest latency in all the sampled intervals. Notice that peak latencies are reduced from more than 2000 cycles in the UD to 600 cycles in the MA-2VC. However, only in the intervals with high traffic rate the increased flexibility offered by the adaptive routing algorithm makes a difference. These intervals are shorter when a higher degree of adaptivity is used in the routing algorithm. As expected, the case for the MA-2VC is



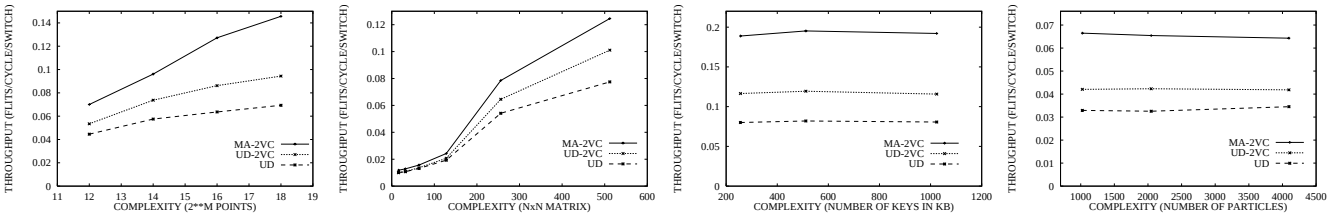
a b c d

Figure 1. Execution times for FFT, LU, RADIX and BARNES



a b c d

Figure 2. Latency for FFT, LU, RADIX and BARNES



a b c d

Figure 3. Throughput for FFT, LU, RADIX and BARNES

the shortest one. On the other hand, intervals with low traffic rate are almost identical regardless of the routing scheme used. A similar behavior is obtained for the Radix application.

We can also see with the Radix application that the network reaches throughput values up to 0.4 flits/cycle/switch, which are higher than the maximum throughput achievable with a uniform distribution. At first glance, these points seem to be saturation points. However, this is not the case. Figure 6 shows the average distance traveled by messages for each time interval with the UD routing algorithm. We can observe that in the high traffic points the average distance decreases, indicating that the traffic pattern is changing to a highly local traffic pattern. On the other hand, we can see that the average distance decreases to less than one, indicating that many messages do not leave the source switch. Thus, the RADIX application does not saturate the interconnection network at these points. However, taking into account that we are using a sequential consistency memory model, the interconnection network could saturate with a more aggressive memory model.

In summary, routing algorithm has a great impact on application execution time in a NOW with a hardware shared memory model. The running applications generate a very specific traffic pattern. This traffic is handled in a different way depending on the routing algorithm implemented within the interconnection network. With minimal adaptive routing, messages cross the interconnection network through minimal paths, leading to reduced latencies. In addition, adaptive routing also allows the use of alternative paths, achieving a more balanced use of resources and increasing network throughput. As a consequence, applications finish earlier.

6 Conclusions

In this paper we analyzed the influence of routing algorithm on the performance of a NOW with a hardware shared memory model using the traffic generated by some real applications. This analysis has been performed by simulating the behavior of networks with irregular topology, using an execution-driven environment. Several SPLASH-2 applications (Barnes-Hut, FFT, Radix, and LU) were used on a distributed shared-memory multiprocessor (DSM) simula-

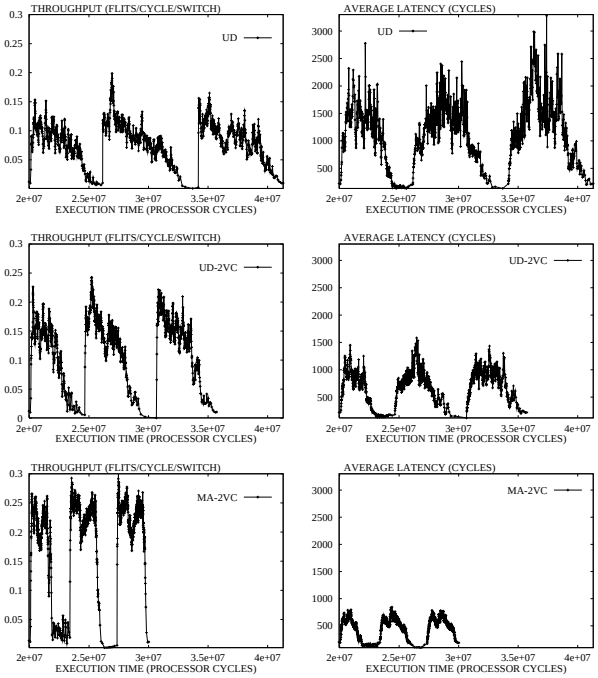


Figure 4. Throughput and latency during the execution of FFT

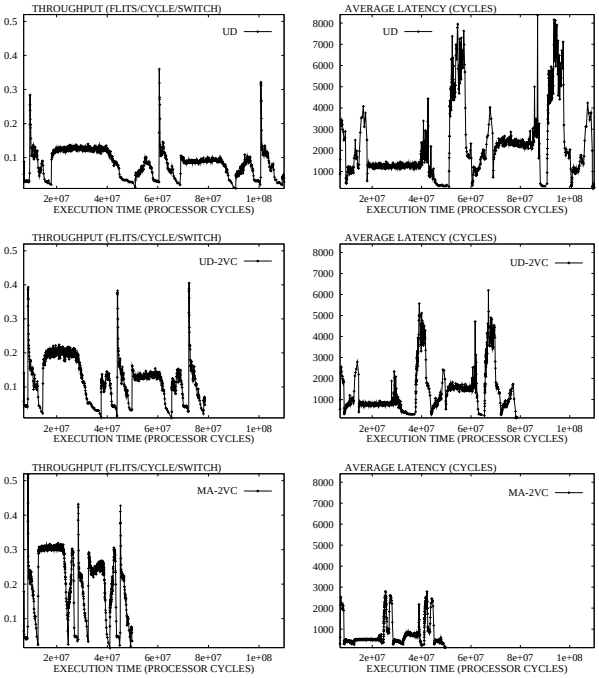


Figure 5. Throughput and latency during the execution of RADIX

tor with 64 processors, a hardware cache-coherence protocol and a network consisting of 16 switches (4 nodes per switch). The execution time of these applications was the main performance measure.

We conclude that the routing algorithm plays a key role in the design of interconnection networks for NOWs using a hardware shared memory model. In particular, when using a sequential consistency model (there are a maximum of p requests per time unit in a system with p processors), the interconnection network reaches high traffic loads in some critical phases of program execution. Hence, efforts must be done in routing algorithm design. These efforts must be oriented toward adding more adaptivity to current routing algorithms like up*/down*. The low latencies and high throughput achieved by more adaptive routing schemes noticeably decrease the overall execution time of the considered applications. Reduction time ranges from 38 % in the worst case to 58 % in the best case. Thus, virtual channels and minimal routing are the key to achieve a significant performance improvement in NOWs systems with a hardware shared memory model.

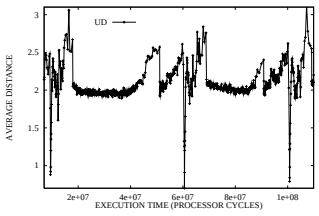


Figure 6. Average distance traveled by messages for RADIX with UD

References

[1] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "TreadMarks: Shared Memory Computing on Networks of Workstations," in *IEEE Computer*, Vol. 29, No. 2, pp. 18-28, February 1996.

[2] W. C. Athas and C. L. Seitz, "Multicomputers: Message-passing concurrent computers," in *IEEE Computer*, vol. 21, no. 8, pp. 9–24, August 1988.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic and W. Su, "Myrinet - A gigabit per second local area network," in *IEEE Micro*, pp. 29–36, February 1995.

[4] G. T. Byrd and M. J. Flynn, "Evaluation of communication mechanisms in invalidate-based shared memory multiprocessors," in *Proceedings of the 1997 Parallel Computer Routing and Communication Workshop*, June 1997.

[5] D. E. Culler and J. P. Singh, *Parallel computer architecture : a hardware-software approach*, Morgan Kaufmann, 1999.

[6] Dolphin, *The Dolphin SCI Interconnect*. Dolphin Interconnect Solutions, `http://www.dolphinics.no`.

[7] J. Duato and P. López, "Performance evaluation of adaptive routing algorithms for k-ary n-cubes," in *Parallel Computer Routing and Communication*, K. Bolding and L. Snyder (ed.), Springer-Verlag, 1994.

[8] J. Duato, S. Yalamanchili and L. M. Ni, *Interconnection Networks: An Engineering Approach*. IEEE Computer Society Press, 1997.

[9] J. Flich, P. López, M.P. Malumbres and J. Duato, "EDINET: An Execution Driven Interconnection Network Simulator for DSM Systems," in *Lecture Notes in Computer Science 1469*, Springer-Verlag, September 1998.

[10] M. Galles, "Spider: A high speed network interconnect," in *IEEE Micro*, vol. 17, no. 1, pp. 34–39, January-February 1997.

[11] D. Garcia, "Servernet II," in *1997 Parallel Computer Routing and Communication Workshop*, June 1997.

[12] K. Gharachorloo, A. Gupta and J. Hennessy. "Performance Evaluation of Memory Consistency Models for Shared-Memory Multiprocessors," in *Technical Report CSL-TR-90-456*, Computer Systems Laboratory, Stanford University.

[13] J. L. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*, Second Edition, Morgan Kaufmann, 1996.

[14] R. Horst, "ServerNet deadlock avoidance and fractahedral topologies," in *Proc. of the Int. Parallel Processing Symp.*, April 1996.

[15] D. Lenoski, et al., "The Stanford DASH multiprocessor," *IEEE Computer*, vol. 25, no. 3, pp. 63–79, March 1992.

[16] D. Magdic, "Limes: A Multiprocessor Simulation Environment," *TCCA Newsletter*, pp 68-71, March 1997.

[17] N. R. McKenzie, K. Bolding, C. Ebeling and L. Snyder, "ChaosLAN: Design and Implementation of a Gigabit LAN Using Chaotic Routing," *1997 Parallel Computing Routing and Communication Workshop*, pp. 211–223. June 1997.

[18] A. G. Nowatzyk, et al., "S-Connect: From networks of workstations to supercomputer performance," in *Proceedings of the 22nd International Symposium on Computer Architecture*, pp. 71–82, June 1995.

[19] W. Qiao and L. M. Ni, "Adaptive routing in irregular networks using cut-through switches," in *Proceedings of the 1996 International Conference on Parallel Processing*, August 1996.

[20] M. D. Schroeder et al., "Autonet: A high-speed, self-configuring local area network using point-to-point links," in *Technical Report SRC research report 59*, DEC, April 1990.

[21] S. L. Scott and J. R. Goodman, "The Impact of Pipelined Channels on k-ary n-Cube Networks," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 1, pp. 2–16, January 1994.

[22] S. L. Scott and G. Thorson, "Optimized routing in the Cray T3D," in *Proceedings of the Workshop on Parallel Computer Routing and Communication*, pp. 281–294, May 1994.

[23] S. L. Scott and G. Thorson, "The Cray T3E networks: adaptive routing in a high performance 3D torus," in *Proceedings of Hot Interconnects IV*, August 1996.

[24] F. Silla, M. P. Malumbres, A. Robles, P. López and J. Duato, "Efficient Adaptive Routing in Networks of Workstations with Irregular Topology," in *Workshop on Communications and Architectural Support for Network-based Parallel Computing*, February 1997.

[25] F. Silla and J. Duato, "On the Use of Virtual Channels in Networks of Workstations with Irregular Topology," in *1997 Parallel Computer Routing and Communication Workshop*, June 1997.

[26] F. Silla and J. Duato, "Improving the Efficiency of Adaptive Routing in Networks with Irregular Topology," in *1997 Int. Conference on High Performance Computing*, December 1997.

[27] F. Silla, M.P. Malumbres, J.Duato, D. Dai, and D.K. Panda, "Impact of Adaptivity on the Behavior of Networks of Workstations under Bursty Traffic," in *Proceedings of the 1998 International Confence on Parallel Processing. IEEE Computer Society*, August 1998.

[28] Speight and J.K. Bennett, "Brazos: A third generation DSM system," in *Proceedings of the 1997 USENIX Windows/NT Workshop*, August, 1997.

[29] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das, "Performance benefits of Virtual Channels and Adaptive Routing: An Application Driven Study," in *International Conference on Supercomputing*, 1997

[30] S. C. Woo et al., "The SPLASH-2 Programs: Characterization and Methodological Considerations," in *Int. Symp. on Computer Architecture*, 1995.