Reconfigurable interconnection networks in Distributed Shared Memory systems: a study on communication patterns

Bui Viet Khoi¹, Pham Doan Tinh¹, Nguyen Nam Quan¹, Iñigo Artudo², Daniel Manjarres², Wim Heirman³, Christof Debaes², Joni Dambre³, Jan Van Campenhout³, Hugo Thienpont²

¹ Department of Electronics and Informatics, Hanoi University of Technology, Viet Nam

² Department of Applied Physics and Photonics, Vrije Universiteit Brussel, Belgium

³ Electronics and Information Systems Department, Universiteit Gent, Belgium

Contact address: bvkhoy@mail.hut.edu.vn

Abstract— The static interconnection network topologies in the distributed shared memory systems (DSM) have several limitations. The reconfigurable interconnection networks may reduce the network congestion, network latency and improve the overall performance. However it is necessary to know when the right moment to perform the reconfiguration is and how to implement it. In this paper, we present our study on communication patterns of parallel scientific and commercial benchmark applications on a simulated but realistic DSM machine and their relation to context switching in the operating system. We also propose a reconfiguration scheme that is triggered by the context switches.

Keywords—reconfiguration; interconnection network; distributed shared memory; multiprocessors; context switch

I. INTRODUCTION

Most of the distributed shared memory multiprocessor systems today have a fixed interconnection network topology connecting the different nodes. These static interconnection network topologies, such as tori and hypercubes, still have connectivity limitations when we scale the systems up. Developing parallel algorithms whose communication patterns closely match a certain interconnection topology of a target parallel system is often a difficult and non-convenient task. And although one interconnection topology may have the ideal node distribution for data interchange for a set of algorithms, it may not be well suited to other set, introducing large latencies on the network for other algorithms [1].

The reconfigurable interconnection networks for multiprocessor systems have attracted a lot of research interests [1-5]. An interconnection network is called reconfigurable if its topology can be changed during program execution, usually to match the communication requirements of an application.

Almost all of the published works on reconfigurable interconnects in the past considered only the abstract models of reconfigurable networks [1,2,4] and addressed the implementation of programmable interconnects for processor arrays by using FPGAs [3,5]. With the advent of new photonic

devices, such as wavelength tunable Vertical Cavity Surface Emitting Lasers (VCSELs), tunable optical filters and resonant cavity detectors, the reconfigurable interconnects can be implemented via optical technologies, with all the extra benefits commonly associated to optical networks in comparison to electrical technologies.

Although several models of reconfigurable interconnection networks for parallel computer architectures have been already proposed, very few of them were constructed and actual performance gain is yet to be demonstrated.

In our research work, we investigate whether the reconfigurable interconnects can improve the performance of distributed shared memory machine via extensive simulations, and we propose a possible implementation making use of available low cost optical technologies.

In this paper, we focus our study on communication patterns of parallel scientific and commercial benchmark applications on a simulated but realistic DSM machine and their relation to context switching in the operating system.

The paper is organized as follows. Section II describes the architecture of the simulated distributed shared-memory machine and the reconfigurable interconnection network that were chosen for this study. The simulation environment used to run the benchmarks and track the communication patterns is described in section III. The results of the simulations and further discussions are presented in section IV. Finally, the conclusions are summarized in section V.

II. SYSTEM ARCHITECTURE

A. Distributed shared-memory multiprocessors

In the distributed shared memory (DSM) architecture, the shared main memory is physically distributed among the processors as local memory units and all processor nodes are connected by an interconnection network. Therefore the memory access can be either local or remote depending on where the data resides. The multiprocessors interconnects found in DSMs can be classified according to the topology connecting the different nodes in the system, each with the

This work was supported in part by the IAP-V 18 PHOTON network sponsored by the Belgian Science Policy office and in part by the VLIR-HUT post-docs Research fund.

processing unit and a part of the distributed global memory.

In the bus-based architectures, the interconnection network is a shared bus that is located between the processor's private caches and the shared main memory. This architecture is also referred to as Symmetric Multiprocessors (SMPs) and has been widely used for small and medium-scale multiprocessors consisting of 2 to 32 processors due to its simplicity. But the shared bus itself becomes a critical architectural bottleneck, as all the data and address transactions sent from the processors and main memory are transmitted through the shared bus, making this traffic saturate the bus for higher node counts. Recently, the shared buses in SMPs computers are being replaced by crossbar switches to handle the physical limitations. However they will continue to have a non-scalable bandwidth limitation since all messages have to be broadcasted to all the processors.

Other topologies different than a bus (mesh, torus, tree...) can make DSM designs very scalable, consisting of hundreds of processors (SGI Origin 2000, HP Convex Exemplar). However they have two main problems: congestion can be found under heavy traffic for some individual links on the network, and communication latency between pairs of nodes can be very high if they are not close together. These problems contribute to increase memory access times and thus slowdown the processor's execution speeds. Hence a reconfigurable interconnection network could alleviate those problems and contribute to improve the performance of DSM computers by dynamically adjusting the network topology to match the run-time communication requirements [6].

B. Reconfigurable interconnection network

The proposed interconnection network architecture for the DSM system in this paper consists of a fixed base network connecting all the nodes (processors and local memories), arranged in a torus topology. In addition, we will place a certain number of free reconfigurable links (see Figure 1) between pairs of nodes that are expected to have a temporal heavy traffic load. These new extra links can be employed as direct point-to-point connections to route the traffic between the busiest processor node pairs in congestion situation, while still keeping the base network for all the other communications. Therefore the network congestion and latency can be reduced significantly. Those direct connections will be alive for a certain interval of time, and then the extra links will be reassigned according to the new traffic measurements.

This network architecture, compared to the other topologies where all links in the network are to be used for the topology reconfiguration, has a number of advantages. It is indeed impossible to disconnect parts of the network when the program is running and it also greatly simplifies the routing and reconfiguration algorithms since the base network will always be available.

Figure 1. Torus topology of the base interconnection network with added reconfigurable links. The numbers correspond to processor node numbers inside the network.

However, to make optimal use of reconfigurable links, it is necessary to know when the right moment to perform the reconfiguration is and where to put these extra links. We want to focus our research on the occurrence of trigger conditions that can lead to a reconfiguration. One of the events on the system triggering the reconfiguration could be the context switches happening on the operating system. Such a context or task switch can be expected to impose high traffic demands on the interconnection network during short intervals of time and may thus offer a possible communication pattern we can use to solve the question of when to reconfigure.

C. Context switching in the Operating system

The Operating system (OS) is a critical part of a multiprocessor system, because it controls the execution of the applications running and the switches in between. As only one task can be executed by one processor at each time, an OS must create structures that enable it to run multiple tasks concurrently. The basic structure for managing the execution of an application is the process; it provides the necessary resources to make each task as an isolated entity. During normal execution, only one process per processor can be executed. After a certain time interval, this processor can switch to another process; this procedure is known as context switch.

By a context switch, the kernel saves the state of the current running process or thread and then loads the state of the next one to be executed. It needs to save enough information about the current execution so that it can be resumed later. Just after the context switch, the processor will work with a completely different set a code and data, therefore the data in the cache will be invalidated and a communication peak to this processor will occur to fill the caches while new read and write instruction are executed.

Before the OS assigns another process to the running state, it must carry out several steps, some of them involving movement of data from different nodes through the interconnection network to the memory. This includes saving

the context of the processor, including program counter and other registers, move the process control block of this process to the appropriate queue in the scheduler, update the process control block of the process selected, update memory-management structures and restore the context of the processor, loading the saved data. All these operations will generate a sudden burst of traffic on the network as these structures are moved from caches and memories.

Hence this suggests that there will be high traffic loads after every context switch, making it an adequate trigger for the adaptation of the network. It motivated us to investigate the communication patterns of different parallel applications on the simulated DSM machine during processes execution.

III. SIMULATION ENVIRONMENT

We have established a full-system simulation environment based on Simics [7], a commercially available execution-driven multiprocessor simulator. Simics is able to simulate complete computing systems, including their operating system and the realistic workloads. The simulator was configured to model a multiprocessor machine based on the Sun Fire 6800 server, with 16 UltraSPARC III processors at 1 GHz running the Solaris 10 operating system. We extended the Simics simulator with an interconnection network module, where we modeled a 4x4 torus network with contention and cut-through routing.

The SPLASH-2 scientific parallel benchmark suite [8], being a standard benchmark for parallel multiprocessors, as well as the Apache web server 2.0 along with the Apache benchmark application, were chosen as the workload applications for stressing the system under test. A statistics module has been developed and added to Simics allowing us to collect all information about network traffic, context switch data, memory and processor information of the executions.

IV. SIMULATION RESULTS

A. Communication patterns and context switches

We have run a set of benchmark programs (FFT, LU, Cholesky, Barnes, Raytrace ...etc.) from the SPLASH-2 application suite with default problem sizes, and also installed the Apache Web server 2.0 on the simulated machine. In order to measure the traffic pattern of benchmarking applications, we have included instructions in the simulator source code so that every time, when a packet is injected to the interconnection network, we catch the name of sender node, receiver, size of the packet and time stamp, and save all this information in a log file.

After running the benchmark programs on the simulated system, we obtain all the information referring to the number of packets that flew through the network, including which processors are their sources and destinations, the delay and the size of every packet during execution of every application. Based on this data, we can calculate the total traffic, including the incoming and outgoing traffic, of every processor on measurement intervals of 100 μ s. The traffic evolution along with the execution time gives us the traffic pattern, or

communication pattern, of an application. For example, Fig. 2 shows the traffic patterns of processor number 1 running the Cholesky algorithm during 20 ms. The total traffic of node 1 is presented in a thick line; the traffic from processor 1 to processor 4 and to processor 6 is given in dotted lines, while context switch happenings are indicated by vertical lines.

We can see that the traffic at processor 1 increases above 1 Gbit/s just after a context switch due to the load/store operations from memory as well as other traffic bursts happening during the context. Besides, the duration of one context is not constant and depends heavily on the scheduling and interruptions happening on the system. Our simulation results also indicate that the traffic from one processor is not divided equally to all other processors. There were always one or two destinations that were getting the majority of the generated traffic during one context and the whole program execution. Those destinations can differ from application to application and at different simulation runs, depend on the algorithm and the scheduling policy of the operating system. For the Apache simulation, the processor 2 and processor 4 got the majority of traffic from processor 1, and total traffic at processor 1 went up to 2 Gbit/s (see Fig. 3).

Figure 2. Traffic patterns at processor 1 running Cholesky 29.O, along with context switch happenings.

Figure 3. Traffic patterns at processor 1 in Apache 2.0, along with context switch happenings

We have made an analysis of traffic contribution of the two busiest nodes to the total traffic at every processor for Cholesky and Apache 2.0. We found that those two nodes represented from 80% to more than 90% of the total traffic of

one processor node (see Fig. 4), and the busiest node could get from 50% - 76% of the total traffic (see Table 1).

Since a large fraction of traffic for one processor is exchanged with one or two other nodes, the addition of one or two extra links should be sufficient to improve the communication performance by diminishing congestion and reducing latency. These findings are important to practically implement a reconfigurable optical interconnect. Due to the physical limitations of optical components, such as tunable lasers and optical tunable filters, and cost limitations, we would only be able to plug one or two extra links at one node.

Figure 4. Traffic contribution of the two busiest nodes in comparison to the total traffic at every processor for Cholesky and Apache 2.0

In our previous works, it was shown that one or two extra reconfigurable links in each node could speed up from 5% to 27% in the execution time for most of the SPLASH-2 applications [9].

TABLE I. THE TRAFFIC CONTRIBUTION OF THE TWO BUSIEST NODES IN APACHE WEB SERVER 2.0

Processor No.	Traffic to the busiest node	Traffic to the two busiest nodes
1	47 %	69 %
2	56 %	81 %
3	59 %	80 %
4	43 %	62 %
5	59 %	83 %
6	60 %	83 %
7	59 %	81 %
8	44 %	66 %
9	58 %	79 %
10	72 %	93 %
11	72 %	87 %
12	73 %	87 %
13	60 %	82 %
14	69 %	88 %
15	76 %	91 %
16	73 %	88 %

We have also developed a module that monitors the occurrence of context switches in the simulated machines and recorded all data relative to the context switches. Although a lot of events can produce a context switch, we are interested only in long process switches belonging to the stressing applications because they involve a lot of data transfers from

one processor to the other processors and main memory, and can lead to congestion situations due to the heavy load on the network. We measured the amount of context switches and the context lengths, i.e. the time duration between context switches, for different applications (see Table 2). The statistic results show that the average context lengths are ranged from 4 ms to 18 ms and one context can last for approximately 1 second in Apache 2.0.

TABLE II. TIME ELAPSED BETWEEN CONTEXT SWITCHES

Application	Context length (ms)		Number of contexts
	Max	Mean	
Barnes	86	18	326
Cholesky 29.O	52	4	324
FFT	27	8	123
LU	20	4	1700
Raytrace	27	11	34
Volrend	25	4	470
Barnes	86	18	326
Apache 2.0	927	17	83

B. Context-switch triggered reconfiguration scheme

Based on the communication patterns and context switches data that we have been measured, we propose a reconfiguration scheme that is triggered by the context switches. In this scheme, we suppose that we can add only one extra link at each node and have access to the context register for determining the context switches from the operating system. At the beginning of a context, a logic unit in each node measures the traffic from that node to all other nodes during a duration Δt and determines the destination node that gets the majority of traffic to place the extra direct link in between. This extra link will live until the next context switch triggers reconfiguration again, and so on.

A distributed algorithm running on every node computes the routes that packets follow and automatically recalculates the routing tables to incorporate or remove the extra links whenever the topology changes.

To be beneficial from reconfiguration, the measurement duration Δt and the time for extra link selection and for switching ($t_{se} + t_{sw}$ as shown in Fig. 5) should be much smaller than the average context length, setting this value to 1 ms for our study.

Figure 5. In every context, the system is monitoring the traffic flow and adjusts its topology

V. CONCLUSIONS

We have shown that there is indeed intensive communication between pairs of nodes after a context switch happens in a distributed shared memory system, and only one or two destination nodes get the majority of those traffics. These results indicate that by making use of a reconfigurable network that can add as much as two extra links at each node, this should be enough to improve the communication and remove congestion and large latencies, resulting in an overall reduction in execution time for parallel application. The switching time for the components required to implement such a reconfigurable network would need to be smaller than 1 ms, which is a value achievable currently for low cost optical technologies.

ACKNOWLEDGMENTS

This work was supported in part by the IAP-V 18 PHOTON network sponsored by the Belgian Science Policy office and in part by the VLIR-HUT post-docs Research fund. Dr. Bui Viet Khoi gratefully acknowledges the receipt of a grant from the Flemish Interuniversity Council for University Development Cooperation (VLIR UOS) which enabled the research team to carry out this work.

REFERENCES

- [1] I. Lee and D. Smitley, "A Synthesis Algorithm for Reconfigurable Interconnection Networks," *IEEE Transaction on Computers*, vol. 37, pp. 691-699, 1988.
- [2] S. Miguet and Y. Robert, "Reduction operations on a distributed memory machine with a reconfigurable interconnection network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3(4), pp. 500-505, 1992.
- [3] T. Sueyoshi, B. O. Apduhan, S. Funakoshi, and I. Arita, "A new approach towards realization of reconfigurable interconnection networks," in *Proceedings of the Eleventh Annual International Phoenix Conference on Computers and Communications*, 1992, pp. 456-463.
- [4] J. L. Sanchez, J. Duato, and J. M. Garcia, "Using channel pipelining in reconfigurable interconnection networks," in *Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing (PDP '98)*, 1998, pp. 120-126.
- [5] L. K. John and E. John, "A dynamically reconfigurable interconnect for array processors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6(1), pp. 150-157, 1998.
- [6] P. Krishnamurthy, "Reconfigurability of the interconnect architecture for chip multiprocessors," in *Proceedings of the 4th International Symposium on Information and Communication Technologies*, 2005, pp. 136-141.
- [7] P. S. Magnusso, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *IEEE Computer magazine*, vol. 35(2), pp. 50 - 58, 2002.
- [8] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and Methodological Considerations," in *Proceedings of the 22nd International symposium on Computer Architecture*, 1995, pp. 24 - 36.
- [9] I. Artundo, L. Desmet, W. Heirman, C. Debaes, J. Dambre, J. M. V. Campenhout, and H. Thienpont, "Selective Optical Broadcast Component for Reconfigurable Multiprocessor Interconnects," *IEEE Journal of Selected Topics in Quantum Electronics* vol. 12(4), pp. 828-837, 2006.