Reconfigurable interconnects in DSM systems: a focus on context switch behavior

I. Artundo[1], D. Manjarres[1], W. Heirman[2], C. Debaes[1], J. Dambre[2], J. Van Campenhout[2], H. Thienpont[1]

1: Department of Applied Physics and Photonics, Vrije Universiteit. Pleinlaan 2, 1050 Brussel, Belgium
{iartundo, dmanjarres, christof.debaes, hthienpo}@tona.vub.ac.be
2: ELIS Department, Universiteit Gent, Sint-Pietersnieuwstraat 41, Gent, Belgium
{wim.heirman, Joni.Dambre, Jan.VanCampenhout}@elis.ugent.be

Abstract. Recent advances in the development of reconfigurable optical interconnect technologies allow for the fabrication of low cost and run-time adaptable interconnects in large distributed shared-memory (DSM) multiprocessor machines. This can allow the use of adaptable interconnection networks that alleviate the huge bottleneck present due to the gap between the processing speed and the memory access time over the network. In this paper we have studied the scheduling of tasks by the kernel of the operating system (OS) and its influence on communication between the processing nodes of the system, focusing on the traffic generated just after a context switch. We aim to use these results as a basis to propose a potential reconfiguration of the network that could provide a significant speedup.
Keywords. Reconfiguration, interconnection network, distributed shared memory, multiprocessors, context switch.

1. Introduction

In DSM multiprocessor machines all the memory of the system is physically distributed among its nodes, and they can access data located in the memory of other nodes in a software transparent way. The interconnection network is thus part of the memory hierarchy and therefore high network latencies cause a significant performance bottleneck in program execution [1]. This situation will become worse in the future, as a result of increasing hardware performance, the rapid growth in instruction level parallelism and the use of multiple process contexts [2]. Reconfigurability in this aspect will allow the system to rearrange the interprocessor communications network to form topologies that are best suited for the particular computing task at hand, allowing for a network topology that closely matches the traffic patterns exhibited by the current application [3].

Optics is a great candidate to introduce fast interconnection networks in the architecture of multiprocessor systems [4]. By using optical interconnects at the scale of the link lengths found in multiprocessor machines, an increase in connectivity and higher communication bandwidths can be achieved, as well as the elimination of frequency dependent cross-talk with galvanic isolation. One important aspect that has not been yet exploited so far is their inherent ability to switch the light paths easily in a data transparent way, paving the way towards adaptable network topologies.

It is necessary to determine the architectural limitations specifically imposed by interconnection networks and propose an efficient way to apply them to multiprocessor machines. For the leading low-cost solutions, a designer must overcome the fact that switching speed and connectivity do not come for granted, so it is necessary to find communication and reconfiguration schemes that not only overpass these limitations but try to use them at their advantage. Communication patterns lasting long enough compared to optical switching times must be found so that we can allow for a slow reconfiguration rate of the links in the topology. The frequent context switches that happen in modern operating systems are the perfect event to be used as a trigger, to establish a regular and profitable reconfiguration on the interprocessor communication network. The peak bandwidth and congestion expected in those intervals will be the key element here.

It is the goal of this work to investigate the communication patterns found on a DSM interprocessor network related to the task scheduling, focusing on the context switches. We will first give an introduction to reconfigurability in interconnection networks. Later on, we will enter in detail on the context switch behavior of the operating system and the mechanisms to use it as a trigger for adapting the network to certain communication patterns. Finally, we will present the results obtained by running full system simulations of real parallel benchmarks and we will discuss them proposing a possible optical implementation of the system.

2. Reconfiguration in Interprocessor Networks

Through reconfiguration the system can adapt the interconnection network to better fit the real-time communication needs, which depend upon the application that is running in the machine. This will alleviate the large bottleneck affecting the current communication networks and moreover can serve as a backup network in case there is a failure in any of the components of the system.

The proposed interconnection network architecture for the DSM system consists of a fixed base network connecting all the nodes (processors and local memories), arranged in a torus topology. In addition, a certain number of freely reconfigurable point-to-point links will be provided (see Fig. 1) between nodes that are expected to have a large communication load. These new links can be used as direct shortcut connections to route the traffic between processor node pairs on the network. This will happen for a certain interval of time, and then the links will be available until a certain interval of time. Afterwards, the extra links will be reassigned according to the new congestion measurements.

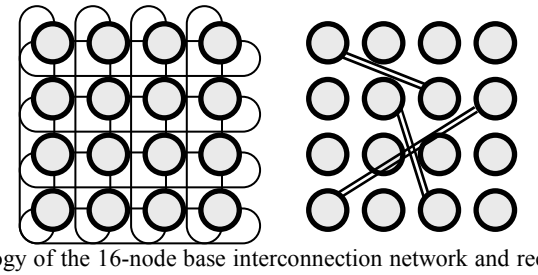


Fig. 1. Torus topology of the 16-node base interconnection network and reconfigurable optical layer with some assigned extra links.

This setup, compared to the case where all links in the network are available to be used for the topology reconfiguration, has a number of advantages because the base network will always be available. It is therefore impossible to disconnect parts of the network, greatly reducing the complexity in the reconfiguration algorithms. However, this reconfigurability can offer advantages to a system considering some requirements are met beforehand.

In this study, we focus our architectural study primarily on the occurrence of events to act as trigger conditions that can lead to a reconfiguration. These events will be the environment switches happening on the OS, expected to impose higher demands on the interconnection network during short intervals of time. The right placement of the extra links in the topology and the implementation of such a network are questions already treated in our previous works [5][6].

3. Context Switching in the Operating System

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*. [7]. The OS used in this work, Solaris 9, has a good support for multiprocessor systems and it is based on a process-thread model where processes are divided in threads in order to be managed by the scheduler [8].

3.1 Scheduling of Processes and Threads, and Temporal Patterns

The OS uses several structures to define each process and thread, like indexed tables or arrays, describing every aspect involved in the process. After a processor has finished its allocated time-slot, a scheduler interrupts its execution, write all relevant processor state information to the memory space pointed by a context register and will pass execution to a next process.

The execution of different tasks is controlled by the scheduler, enabling processes and threads to work on one system by switching constantly between them on a short time interval. Previous works [9] have demonstrated the relationship between task scheduling and the end-point or network contention in dynamical interconnections, proposing new scheduling models that could be used to be aware of the communication layer state.

The process scheduler of Solaris is developed in a multilayer process-thread model. Solaris sets fixed time-slices that range between 20 and 200 ms for the lowest priorities in the system, that are the threads belonging to user processes. The above means that it can be expected that the interprocessor network will be under heavy load with communication peaks at predetermined intervals of time, when the switches between processes are happening. Overall, these intervals are in no way regular along time, because the execution is always being interrupted and continued. However, intervals in the order of tens or even hundreds of milliseconds will be long enough to be profitable for reconfiguration, even for slow switching technologies that can prepare and adapt the network to the expected burst of incoming traffic due to these process/context switches.

3.2 Reconfiguration through the Context Switches

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. Just after the context switch, the processor will work with a completely different set of 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. All these operations will generate a sudden burst of traffic on the network as these structures are moved from caches and memories.

An OS that can make use of a reconfigurable interconnection network will need to track every context switch and keep record of the traffic patterns it generates. It will be able then to inform the interconnection hardware when and how a reconfiguration can take place. As there is no practical way to predict in advance the occupancy of the network due to the new traffic and its destination, the system will determine to which node most of the traffic was flowing last time the same context was run, and prepare the network in consequence for this expected increase of load. The reconfiguration would then be triggered always by a context switch. The performance gain obtained will be optimum in case the network reconfiguration fits the expected traffic to a certain destination after a context switch.

4. Simulation Environment

For studying all the aspects involved in these contexts switches and build a coherent reconfiguration architecture, we have established a full-system simulation environment based on the commercially available Simics simulator [10]. A more detailed description of our environment can be found in [11].

The interconnection network is a custom extension to Simics, where we modeled a 4x4 torus network with contention and cut-through routing. In our simulations, only two multithreaded benchmark applications were strictly run at the same time as the main load, so we can suppose with a high level of certainty that on a context switch we will switch between the benchmarking applications and the daemons of the kernel (around 10-15). We focus our results on two types of loads: in one case we have loaded the machines with two simultaneous runs of a multithreaded Barnes algorithm from the SPLASH-2 scientific parallel benchmark suite [12], doing as well other simulations with several applications of the same suit for comparison purposes (FFT, Radiosity, etc.), and secondly the Apache web server v.1.3 concurrently run with the SURGE request generator [13]. Each of the above user process will start 16 threads, so that at all time as much as 32 threads will be competing to be run on the 16 processors of the machine. Since the proposed reconfiguration scheme performance scales with the number of threads and processing nodes, our simulation results will benefit from higher processor counts. However, due to the extremely long simulation times and routing complexities, it was not feasible for the moment to perform simulations with more than 16 processors.

Interrupts and system calls are managed by the OS in the machine, and in most cases do not require a whole process switch, so the context switches produced by them tend to be short and with low communication rates. We will not take them into account since their characteristics (in length and bandwidth consumption) did not offer a proper base to be used by a possible reconfiguration trigger. When filtering these short interrupts and system calls, we have only used execution intervals on every node lasting at least 10 ms, reconfiguring this way the whole network faster than the smallest 20 ms time slice executed on a single processor without interruption.

5. Evaluating Communication and Reconfiguration

In this section we present a study of the dynamics of the context switches happening on the system, and show how they can be used as a reconfiguration trigger. To show a preliminary effect of such reconfiguration, we run the simulations again, this time enhanced by the extra links placed between several pairs of nodes that are expected to have a high communication load due to a context switch happening on the OS.

5.1 Context Switch Communication Patterns

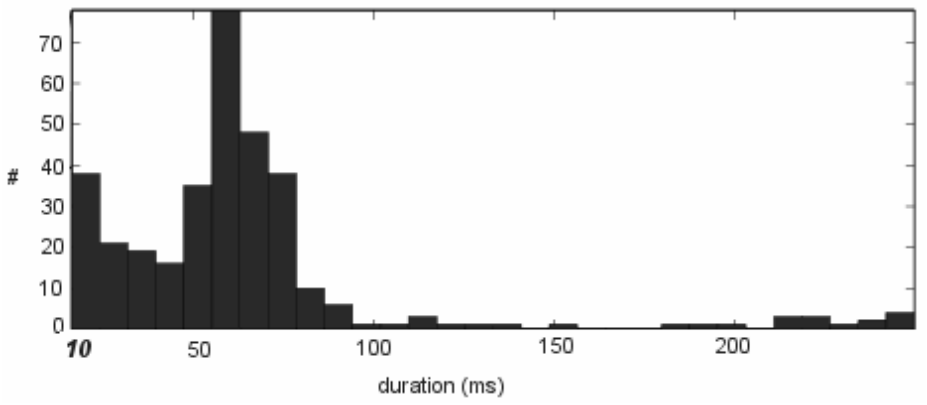
Within Simics, we have developed a module that monitors the occurrence of context switches in the simulated machine. From the possible events that can produce a context switch, we are mainly interested in process switches because they involve more of interchanged due to cache invalidation. In Table 1, values related to the average and maximum lengths of the contexts, as well as the number of switches, are presented during a 1400 ms benchmark execution.

Table 1. Time elapsed between several context switches

	Length (ms)		
Application	Mean	Max	# Switches
SPLASH-2: FFT	9.94	12.96	303
SPLASH-2: Cholesky	7.89	13.35	3402
SPLASH-2: Ocean	10.35	14.10	3637
SPLASH-2: Radiosity	10.48	14.28	133
SPLASH-2: Barnes	14.42	224.236	111
Apache Web server	86.58	1119.081	115

The different behavior for every application and their interaction with the OS can be clearly seen here in the number of context switches occurred during simulation. Cholesky and Ocean were the more multithreaded parallel applications, originating much more switches on the system than the other ones. A large variation in the lengths can be observed for Barnes since mean and maximum values are much separated one from each other. We have plotted in Fig. 2 histograms of the context durations for the execution of the Barnes algorithm and the Apache web server.

(a) Distribution of the contexts according to their duration (Barnes)



(b) Distribution of the contexts according to their duration (Apache)

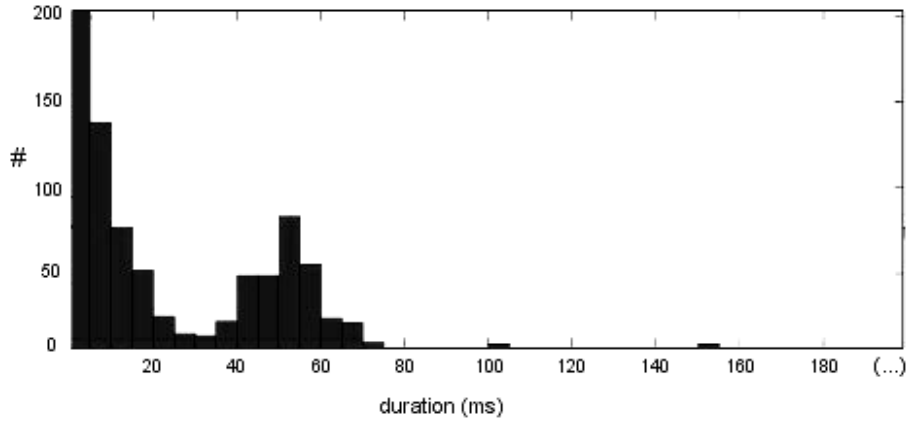


Fig. 2. Histogram distribution of the length of the contexts for Barnes and Apache. More than 1300 contexts last for less than 10 ms, and we can clearly observe a second peak of contexts around 50 ms.

Despite the fact that we have already filtered the context switches with length not enough to be considered profitable to trigger a reconfiguration (< 1 ms), the majority of them have a short duration. It is remarkable that the number of longer contexts is still significant, taking into account that the simulation time was less than 1.5 seconds.

With Apache, the context lengths are considerably longer, with some contexts lasting for even more than one second. While in the Barnes simulations every node was sharing data and a lot of interaction occurred, in the Apache simulation the different concurrent processes are more independent and run during longer time intervals. The process that contains the Apache's kernel will receive the largest lump of traffic and will provide the requested pages by the other nodes, resulting in a large amount of interprocessor communication on this node.

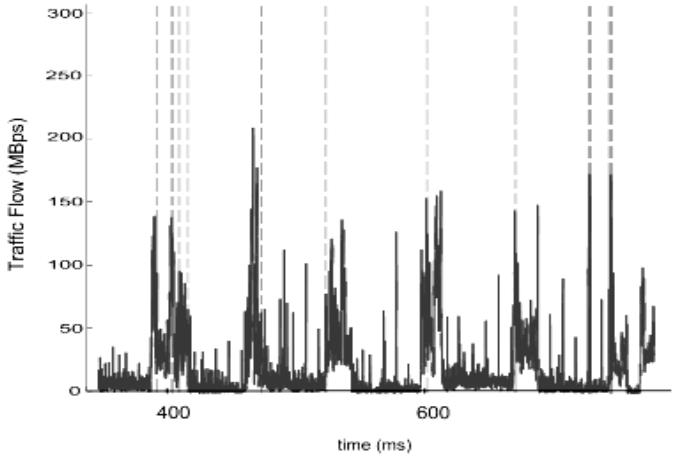


Fig. 3. Detail of outgoing traffic observed in a single node of the system during simulated time. Dashed lines are shown when a context switch occurs.

We show how the generated traffic is correlated with the context switches. In Fig. 3 the traffic flow of one single node is presented while context switches are indicated by vertical lines. At first glance, we can see how just after every context switch the bandwidth consumed on the network increases due to load/store operations from memories. This sudden rise, compared to the mean bandwidth consumed during the rest of the execution, is what we will consider a traffic burst. In some cases, we can even observe a peak of bandwidth consumption just before a context switch is happening, or even when no context switches are happening at all. This means that also other bursts of traffic are generated by the running application.

Next we focus on the time length of these bursts. Hereto, we first define exactly how the length of the burst is measured, i.e. the burst duration is the time difference between the moment of maximum bandwidth of the burst and the moment when traffic drops to 10% of that maximum (see Fig. 4). If we define $T_{k,l}(t)$ as the instant traffic flowing from node k to node l at time t, and τ^i_k the time where context switch i is happening at node k, we have that the bursts of traffic happening just after a context switch will be represented by:

$$B^i_{k,l}(t - \tau^i_k) = \Sigma T_{k,l}(t) \quad . \quad \tau^i_k \leq t < \tau^{i+1}_k \qquad \textbf{(1)}$$

The amount of traffic moved by a burst is therefore directly related to the burst length. However, this traffic measured on one node can go to or arrive from different destinations. The proposed reconfiguration scheme would only add one extra link to a pair of nodes and hence it is needed to predict which node pair is going to show the highest traffic for a next context switch. In a low-latency interconnection with small transmission buffers, this can lead to congestion that can worsen the situation.

After a context switch there was always one destination that was getting the majority of generated traffic, usually with a bandwidth that was 3-4 times higher than averaged traffic to other destinations. It will be critical to accurately know the final destination of this majority of this data communication in order to rearrange the topology and set the new extra link to the proper end node.

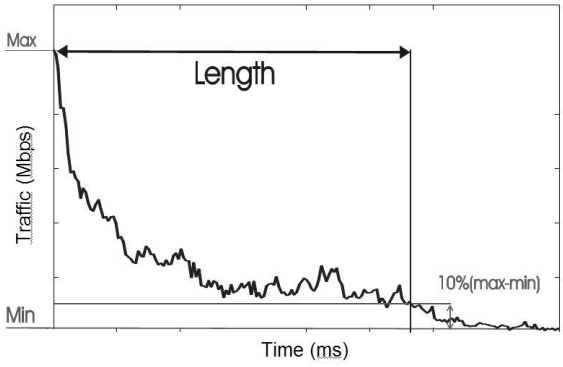


Fig. 4. Diagram of a traffic burst generated after a context switch.

5.2 Context-switch Triggered Reconfiguration

Once the behavior of the communication system was monitored and measured, we have a base to establish the reconfiguration scheme that is triggered by the context switches. Reassigning dynamically the extra links to different node pairs with the higher instant load is expected to result in a speedup of the application running on the system. No limits are imposed on which 16 node pairs are connected at each time (the results of adding more realistic constraints can be found in in [5][6]). Therefore, the 16 busiest node pairs in every reconfiguration interval can be directly connected by extra links according to measurements done on the previous reconfiguration interval.

This way, for the last part of this study we implemented a basic reconfiguration scheme. For this preliminary study on the effects of reconfiguration we had no insight on the scheduler of the Solaris OS, and therefore could not use any information on the prediction for when and to which process a processor will switch. To get however some insight into the excepted performance speed-up, we partitioned the simulated time in discrete reconfiguration intervals such that the topology changes take place at certain moments (see Fig. 7). These intervals should be long enough to amortize on the temporal cost of reconfiguration, during which the extra links are being repositioned and are unusable. The trigger event for a new reconfiguration would be a context switch happening on the system, and the length would be that of the new context. Of course, a prediction model is needed to adapt the network for the upcoming switch, as it is unknown *a priori* when a switch will happen.

As a basic prediction model, previously described in [14], we have divided the simulation time in reconfiguration intervals t_{reconf}. For now, we have not considered any down-time (due to extra link selection and optical switching, $t_{se}+t_{sw}$ as shown in Fig. 5) that occurs during network readjustments to keep the performance study independent of the chosen switching technology. As long as the reconfiguration interval is chosen to be significantly longer than both, this is a good approximation. We have furthermore assumed equal characteristics for the extra links and the base network links, yielding the same average per-hop packet latency for both types of links. The destination node of the extra link will be that measured to have the largest bandwidth consumption on previous reconfiguration intervals.

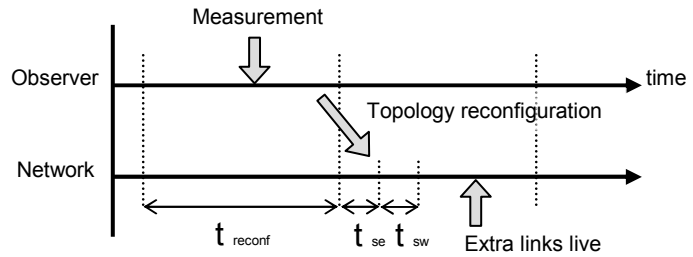


Fig. 5. In every reconfiguration interval, the system is monitoring the traffic flow, such that it can adjust the topology to accommodate the expected communication needs after a context switch.

These connections are established just before a relevant context switch is expected. Of course, this will always be restricted to the prediction model used for determining the occurrence of a switch to a certain context and the destination of most of the traffic generated for that event. As computer communication is basically

unpredictable, it is necessary to constantly monitor the communication flow on the network and extract valuable information on the detected traffic patterns, incorporating it into a prediction model that will do the reconfiguration job.

We proceeded with an implementation based on the accesses to the context register for determining the switches from the OS. There was a certain level of noise (2-5%) on application runtimes, stemming from the initial state of the cache memories as well as other scheduled internal tasks of the OS at the beginning of the simulations. In a real life case, we will not have perfect prediction of the context switches, and there will be a slight time shift between prediction and actual occurrence.

After adding the extra links on reconfiguration intervals triggered by context switches of no less than 100 μs, latency was greatly reduced for a large percentage of the traffic, and the base network was relieved so that less congestion occurred. We found speed-ups in the overall execution time between 8-11% for most of the SPLASH-2 applications. This can be translated into a larger improvement in communication latency that better reflects the performance gain directly obtained by the reconfiguration. Further work is still undergoing to more accurately implement the reconfiguration and obtain a better performance. A more pronounced gain is expected with a more precise prediction of the moment the context switches are happening and of the final destination of the bursts generated. Simulating also larger networks will lead to higher savings in hop distances between nodes. Future work will include expanding the prediction model to more accurately follow the congestion on the network caused by different factors, and not only limited to context switches.

6. Reconfigurable Optical Network Implementation

Our proposal to build the reconfiguration layer of the interconnection network would consist of a tunable optical transmitter per processor node, transmitting data on a fixed number of source wavelengths. For scalability issues it would also be desirable to implement a design that allows the inclusion of new sets of processors as the size of the network increases, via optical broadcasting of the light in several subsets of nodes. Each processor node would also incorporate an optical receiver which is sensitive to one wavelength only. Hence, by tuning the wavelength of each transmitter one would address the destination. More on this proposed optical implementation and the optical broadcasting can be found in [15].

7. Conclusions

The context switch offers a recurrent event that can be used as a base to predict high periods of heavy load in the internal communication of the machine. We can conclude that there are indeed clear intervals corresponding to switches leading to periods of high communication between the nodes of the system. In many cases, the presence of these bursts is overlapped with peaks of traffic coming from the normal execution process and a reconfiguration which takes place on this moment can take profit for the whole context traffic.

However the observed variability of the context switch durations in this study requires more attention in distinguishing and predicting context switches by the operating system. We briefly showed the possibility of using these traffic bursts by a reconfigurable network that is able to modify its topology over the time, obtaining a first significant speed up around 10% in the overall execution time. This interconnect

would be possible to implement by current slow, low-cost optical switching technologies.

References

1. Dai, D., Panda, D.K.: How Much Does Network Contention Affect Distributed Shared Memory Performance?, Proc. of the Int. Conf. on Parallel Processing, (1997) 454-461
2. Krewell, K.: Best servers of 2004: where multicore is the norm, Microprocessor report, January (2005)
3. Krishnamurthy, P.: "Reconfigurability of the interconnect architecture for chip multiprocessors", Proc. of the 4th International Symposium on Information and Communication Technologies, (2005) 136–141
4. Mohammed, E. et al.: "Optical interconnect system integration for ultra-short-reach application," Intel Technology Journal, Vol. 8, Num. 2, (2004)
5. Heirman, W., Artundo, I., Desmet, L., Dambre, J., Debaes, C., Thienpont, H., Van Campenhout, J.: "Speeding up multiprocessor machines with reconfigurable optical interconnects", Proc. of SPIE, Optoelectronic Integrated Circuits VIII, Vol. 6124 (2006) 156-167
6. Artundo, I., Desmet, L., Heirman, W., Debaes, C., Dambre, J., Van Campenhout, J., Thienpont, H.: Selective optical broadcasting in reconfigurable multiprocessor interconnects, Proc. of SPIE Photonics Europe, Vol. 6185, (2006)
7. Tanenbaum, A.S.: Modern Operating Systems 2nd ed., ISBN 0130313580, Prentice Hall, (2001)
8. Multithreading in the Solaris Operating System, Sun Microsystems technical whitepaper, (2002)
9. Sinnen, O., Sousa, L.A.: Communication contention in task scheduling, IEEE Transactions on Parallel and Distributed Systems, Vol. 16, (2005) 503-515
10. Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hållberg, G., Högberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A Full System Simulation Platform, IEEE Computer, (2002) 50-58
11. Heirman, W., Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., Van Campenhout, J.: Predicting Reconfigurable Interconnect Performance in Distributed Shared-Memory Systems. Integration, the VLSI Journal: Special Issue on System Level Interconnect Prediction (to appear), (2006)
12. Woo, S., Ohara, M., Torrie, E., Singh, J., Gupta, A.: The SPLASH-2 programs: characterization and methodological considerations, Proc. of the 22nd Annual International Symposium on Computer Architecture, (1995) 24-36
13. Barford, P., Crovella, M.: Generating representative web workloads for network and server peformance evaluation, Proc. ACM SIGMETRICS, (1998) 151-160
14. Heirman, W., Dambre, J., Van Campenhout, J.: Congestion Modeling for Reconfigurable Inter-Processor Networks, Proc. of the International Workshop on System Level Interconnect Prediction, (2006) 59-66
15. Artundo, I., Desmet, L., Heirman, W., Debaes, C., Dambre, J., Van Campenhout, J., Thienpont, H.: Selective Optical Broadcast Component for Reconfigurable Multiprocessor Interconnects, Journal on Selected Topics in Quantum Electronics: Special Issue on Optical Communications (to appear), (2006)