Available online at www.sciencedirect.com
ScienceDirect
INTEGRATION, the VLSI journal 40 (2007) 382–393
INTEGRATION the VLSI journal
www.elsevier.com/locate/vlsi

Predicting reconfigurable interconnect performance in distributed shared-memory systems

W. Heirman[a,*], J. Dambre[a], I. Artundo[b], C. Debaes[b], H. Thienpont[b], D. Stroobandt[a], J. Van Campenhout[a]

[a]*ELIS Department, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium*
[b]*TONA Department, Free University of Brussels, Pleinlaan 2, 1050 Brussels, Belgium*

Received 28 October 2005; received in revised form 9 May 2006; accepted 11 May 2006

Abstract

Reconfigurable interconnection networks have been shown to benefit performance in distributed shared-memory multiprocessor machines. Usually, performance measurements for these networks require large numbers of slow full-system simulations, making design-space exploration a cumbersome and time-consuming task. In this paper, we present a prediction model for the performance of a reconfigurable network, based on a single full-system simulation and a much shorter, per parameter set post-processing phase. We provide simulation results establishing the relative accuracy of the technique and analyze the impact of several assumptions that were made. With our method, a quick evaluation of a large range of parameters is now possible, allowing the designer to make well-founded design trade-offs.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Prediction model; Interconnection network; Reconfiguration; Distributed shared-memory

1. Introduction

Electrical interconnection networks connecting the different processors and memory modules in a modern large-scale multiprocessor machine are running into several physical limitations [1]. In shared-memory machines, where the network is part of the memory hierarchy [2], the ability to overlap memory access times with useful computation is severely limited by inter-instruction dependencies. Hence, a network with high latencies causes a significant performance bottleneck.

It has been shown that optical interconnection technologies can alleviate this bottleneck [3,4]. Mostly unhindered by crosstalk, attenuation and increased capacitive busload, these technologies will soon provide a cheaper, faster and smaller alternative to electrical interconnections, on distances from a few centimeters upward. Massively parallel inter-chip optical interconnects [5–7] are already making the transition from lab-settings to commercial products.

Optical signals may provide another advantage: the optical pathway can be influenced by components like steerable mirrors, liquid crystals or diffractive elements. In combination with tunable lasers or photodetectors these components will enable a runtime reconfigurable interconnection network [8,9] that supports a much higher bandwidth than what is achievable through electrical reconfiguration technology. From a viewpoint higher in the system hierarchy, this would allow us to redistribute bandwidth or alter the network topology such that node-pairs that communicate intensely have a direct high-bandwidth, low-latency connection.

However, the switching time for most of these components is such that reconfiguration will necessarily take place on a time scale that is significantly above that of the individual memory accesses. The efficiency with which such networks can be deployed will therefore strongly depend on the temporal behavior of the interprocess data transfer

*Corresponding author. Tel.: +32 9 264 95 27; fax: +32 9 264 35 94.
E-mail addresses: wim.heirman@elis.ugent.be (W. Heirman), iartundo@tona.vub.ac.be (I. Artundo).

patterns. We have characterized the locality in both time and space of the traffic flowing over the network in [10], using large-scale simulations of the execution of real benchmark programs with a simulation platform based on the Simics multiprocessor simulator [11]. We have found that long periods of intense communication occur between node pairs suggesting that slowly reconfiguring networks can result in a significant application speedup. Subsequently, we have included the model of a specific reconfigurable network in our simulator, enabling us to measure the attained speedup [8].

When designing the interconnection network for a new line of machines, one would typically like to simulate the speedup of a number of benchmark applications for a range of network parameters, allowing the designer to make the right trade-offs. This can easily require thousands of simulation points. In a *full-system simulation*, the full machine is modeled, including processors, caches, memories and the interconnection network, allowing the traffic on the network to be driven by the actual benchmark application. However, one such simulation can take up to several days to complete, so it is impractical, or even impossible, to do a full-system simulation for each benchmark application and each set of network parameters. The typical solution for this problem is not to employ full-system simulation but to only model the interconnection network. The network traffic is now no longer generated by an actual parallel application, but by a statistical traffic generator [12]. These traffic generators are usually good for modeling simple traffic such as uniform distributions or broadcasting behavior, which can suffice to evaluate static networks. The reconfigurable networks we propose, however, depend on low-frequency dynamics of the network traffic such as bursts, which are not sufficiently modeled in existing traffic generators.

Rather than trying to incorporate complicated dynamics into existing traffic generators, we choose to develop a new technique that takes an actual traffic trace as its input, as generated by one full-system simulation. We presented this technique in [13]. In the present paper, we extend our prediction model to handle network implementations that require a more complex extra link selection algorithm, and apply it to evaluate a reconfigurable network with more realistic topology parameters. We also show simulations for a much wider range of network parameters, showing the high relative accuracy of our technique.

Finally, note that although most of the technological arguments given in this paper are aimed towards an implementation of the reconfigurable network using optics, the application of our prediction model is not limited to optical networks. Any technology that can implement a run-time adaptable topology that fits our general network architecture can be evaluated using this technique.

In Section 2, we describe in more detail the architecture of both the shared-memory machine and the reconfigurable network that were used in this study. Section 3 gives the methodology that was followed to obtain the communication patterns. The prediction model is presented in Section 4. Section 5 gives the prediction results and compares them with the actual speedup. We also introduce some improvements in our network architecture and show that the prediction method can be adjusted accordingly. In Section 6, some future work is discussed. The conclusions are summarized in Section 7.

2. System architecture

2.1. Multiprocessor architecture

Multiprocessor machines come in two basic flavors: those that have a tight coupling between the different processors and those with a more loose coupling. Both types can conceptually be described as consisting of a number of nodes, each containing a processor, some memory and a network interface, and a network connecting the different nodes (Fig. 1). In the extreme end of the loosely coupled family we find examples such as the *Beowulf cluster* [14], in which the network consists of a commodity technology such as Ethernet. This simplistic interconnection network, connected to the processors through several layers of I/O-busses, results in relatively low throughput (1 Gbps per processor) and high latency (several hundred microseconds). These machines are necessarily programmed using the message passing paradigm, and place a high burden on the programmer to efficiently schedule computation and communication.

Tightly coupled machines usually have proprietary interconnection technologies that are situated at an architectural level that is much closer to the processor, resulting in much higher throughput (tens of Gbps per processor) and very low latency (down to a few hundred nanoseconds). This makes them suitable for solving problems that can only be parallelized into tightly coupled subproblems (i.e., that communicate often). It also allows them to implement a hardware-based shared-memory model, in which communication is initiated when a processor tries to access a word in memory that is not on the local node, *without programmer's intervention*. This makes shared-memory-based machines relatively easy to program. Since the network is now part of the memory

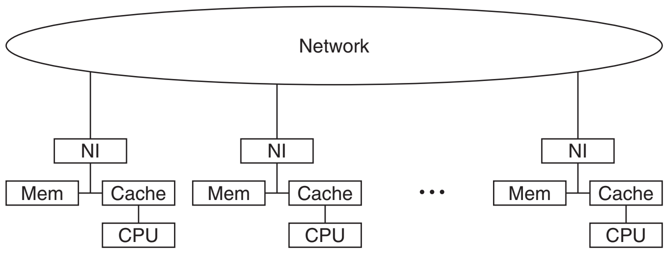


Fig. 1. Schematic overview of a multiprocessor machine. For message-passing machines, the network traffic is under control of the application. In shared-memory machines, network traffic is generated by the network interfaces (NI) in response to non-local memory accesses by a processor.

hierarchy, it also makes such machines much more vulnerable to increased network latencies.

Modern examples of the latter class of machines range from small, 2- or 4-way SMP server machines (including dual-core processors where both CPUs are on the same silicon chip), over mainframes with tens of processors (Sun Fire, IBM iSeries), up to supercomputers with hundreds of processors (SGI Altix, Cray X1). The larger types of these machines are already interconnect limited, and since the capabilities of electrical networks are evolving much more slowly than processor speeds, they make very likely candidates for the application of reconfigurable optical interconnection networks.

For this study, we consider a machine in which coherency is maintained through a directory-based coherency protocol. This protocol was pioneered in the Stanford DASH multiprocessor [2], and is, in one of its variants, used in all modern large shared-memory machines. In this computing model, every processor can address all memory in the system. Accesses to words that are allocated on the same node as the processor go directly to local memory, accesses to other words are intercepted by the network interface. This interface will generate the necessary network packets requesting the corresponding word from its home node. Since processors are allowed to keep a copy of remote words in their own caches, a cache coherency protocol has to be implemented. The network interfaces keep a directory of which processor has which word in its cache, and make sure that, before a processor is allowed to write to a word, all copies of the same word in the caches of other processors are invalidated. Network traffic thus consists of both coherency-related traffic (control packets such as invalidate requests) and data traffic (words that were not in a cache due to cold, conflict, capacity or coherency misses). Therefore, one memory access can take the time of several network round trips (hundreds of nanoseconds). This is much more than the time that out-of-order processors can occupy with other, non-dependent instructions, but not enough for the operating system to schedule another thread (simultaneous multithreading (SMT) can help in this case, but has very limited gain in practice). This makes it very difficult to effectively hide the communication latency, and makes the system performance very much dependent on network latency.

2.2. A simple reconfigurable network architecture

Previous studies concerning reconfigurable networks have mainly dealt with fixed topologies (usually a mesh or a hypercube) that allowed swapping of node pairs, incrementally evolving the network to a state in which processors that often communicate are in neighboring positions [15,16]. However, algorithms to determine the placement of processors turned out to converge slowly, or not at all when the characteristics of the network traffic change rapidly.

Therefore, we assume a different and more modest network architecture in this study. We start from a base network with fixed topology. In addition, we provide a second network that can realize a limited number of connections between arbitrary node pairs—these will be referred to as *extra links* or *elinks*. A schematic overview is given in Fig. 2. To simplify routing the elinks are used exclusively by the two linked nodes, multihop transfers use only the base network.

An advantage of this setup, compared to other topologies that allow for more general reconfiguration, is that the base network is always available. This is most important during periods where the extra network is undergoing reconfiguration and may not be usable (Fig. 3). Routing and reconfiguration decisions are also simplified because it is not possible to completely disconnect a node from the others—all nodes are at all times connected through the base network.

To make optimal use of the extra connections, they should speed up memory accesses that are in the critical path of the application. Since it is very hard, if not impossible, to determine which accesses are in the critical path of any given application, we place the elinks between the node pairs where communication is the most intense (measured in bytes transferred per fixed-length time interval). This way, congestion—and the resulting latency—can be avoided, and a large fraction of the traffic, hopefully including most of the critical accesses, can be given a single-hop pathway, minimizing routing and arbitration delays and resulting in the lowest possible

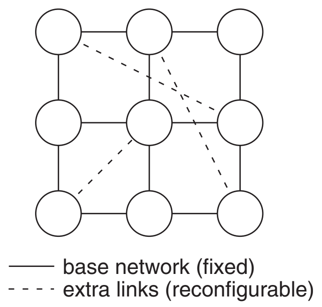


Fig. 2. Reconfigurable network topology. The network consists of a base network, augmented with a limited number of direct, reconfigurable links.

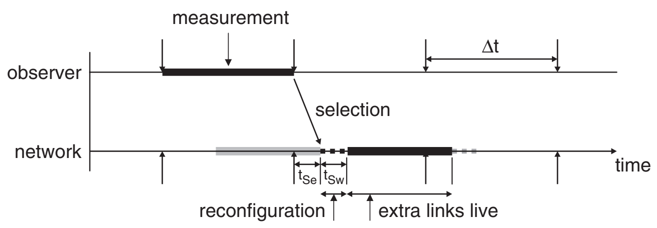


Fig. 3. The observer measures network traffic, and after each interval of length Δt makes a decision where to place the extra links. This calculation takes an amount of time called the *selection time* (t_{Se}). During the *switching time* (t_{Sw}), reconfiguration will take place making the extra links temporarily unusable.

latency. The remaining traffic will use the base network, possibly being routed over several intermediate nodes, and hence will experience a higher latency.

Since the network traffic changes over time, the node pairs with the most intense communication will change so we need to revise the position of the elinks over time. Therefore, we reconfigure the network at specific intervals, the length of each interval being a (fixed) parameter of the network architecture (the 'reconfiguration interval', denoted by Δt). Traffic is observed by a reconfiguration entity during the course of an interval, and total traffic between each node pair is computed. At the end of the interval the new positions of the elinks are determined and the network configuration is updated accordingly. Computing the new configuration takes time, the *selection time* (t_{Se}). Its duration depends on the complexity of the selection algorithm and the method of implementation (in hardware or software). In addition, the reconfiguration itself is not immediate: depending on the technology, reconfiguration can take from 100 μs up to several ms, this is the *switching time* (t_{Sw}). Therefore, the reconfiguration interval Δt will be one of the most important parameters. It should be long enough to amortize on the cost of reconfiguration, during which the elinks are unusable, but it must be sufficiently short to keep pace with the changing demands made by the application. Being able to quickly evaluate different reconfiguration times (and other network parameters), which is made possible by the technique described in this paper, enables a network designer to make the right trade-offs.

2.3. Implementation

The physical implementation of the reconfigurable optical network we envision can be done by using low-cost tunable laser sources, a broadcast-and-select scheme for providing the extra optical links, and wavelength selective receivers on every node (Fig. 4). For the transmission side, vertical cavity surface emitting lasers (VCSELs) are preferred for their low power consumption, easy array integration and coupling into optical fibers. Their tuning range (a few tens of channels) and speed (between 100 μs and 10 ms) is adequate for following the traffic patterns targeted in this study. The broadcasting can be done through the use of a starcoupler-like element that reaches all the nodes. By tuning the laser source, the right destination is addressed. When scaling up to tens of nodes or more this is no longer feasible: the number of available wavelengths is finite, also such a wide broadcast would waste too much of the transmitted power. In this case, a component like a diffractive optical prism can be used, which broadcasts light from each node to its own subset of receiving nodes. On the receiving side, resonant cavity photodetectors (RCPDs) make each node susceptible to just one wavelength. Integration of all these optical components has been proven and optical interconnects are currently arriving to the midrange servers. More

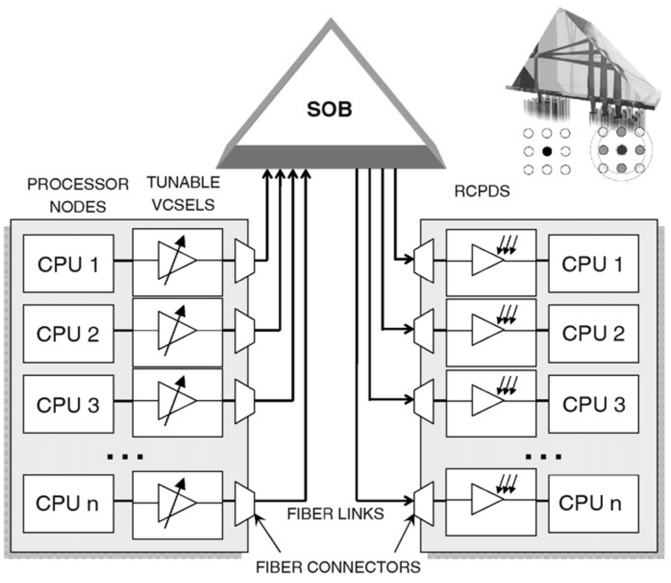


Fig. 4. Schematic representation of the complete reconfigurable optical interconnect. A processor node transmits data on one of nine wavelengths $\lambda_1 \ldots \lambda_9$. The selective optical broadcast element (SOB) distributes the signal towards nine fellow processor nodes. Since every receiving processor node is sensitive to one wavelength only, the target processor node is selected by emitting at the appropriate wavelength.

information about this envisioned implementation can be found in [8].

The simple network model from Section 2.2 provides us with a unified, parameterized architecture that allows us to analyze and simulate several types of implementations. A realistic implementation like the one just described will, of course, impose some limitations. For instance, the selective optical broadcast device only allows a subset of all node pairs to be directly connected with an elink. However, for the current study we do not yet take these limitations into account. Extending our work of [13], we did impose a limitation on the number of elinks that can connect to a node, here called the *fanout*. This restriction can be caused by technological constraints such as a limited off-chip bandwidth, or as a result of reducing cost by limiting the number of optical transceivers per node. For instance, the implementation depicted in Fig. 4 has only one tunable transmitter and one selective receiver per node, restricting the fanout to one.[1]

Another, non-optical implementation of this general architecture might be a network that is part packet-switched and part circuit-switched. The base network is made up by the packet-switched part, for node pairs that communicate often a circuit can be set up. This provides a guaranteed, low-latency path for traffic between both

[1]An optical interconnection (light source $\rightarrow$ waveguide $\rightarrow$ detector) is unidirectional. Elinks as defined in this work are bidirectional and therefore consist of two such assemblies, but each elink still counts as only one when computing the fanout. In theory, it is not necessary for the elinks to be bidirectional. However, since the implementation of a shared-memory model uses a request–response protocol it is not considered very useful to speed up the request but not the response or vice versa.

nodes, allowing them to bypass arbitration and buffering in the intermediate nodes. Setting up a circuit requires several packet round-trips, a circuit should therefore stay in place for a 'longer' period of time to amortize on this initial cost (with 'long' meaning a period spanning multiple memory accesses). The circuit setup time is represented by the 'switching time' in our general architecture.

3. Methodology

We have based our simulation platform on the commercially available Simics simulator [11]. It was configured to simulate a multiprocessor machine resembling the Sun Fire 6800 server, with 16 UltraSPARC III processors clocked at 1 GHz and running the Solaris 9 operating system. Stall times for caches and main memory are set to realistic values (two cycles access time for L1 caches, 19 cycles for L2 and 100 cycles for SDRAM). The directory-based coherency controllers and the interconnection network are custom extensions to Simics, and model a full bit vector directory-based MSI-protocol and a packet-switched 4×4 torus network with contention and cut-through routing. For the simulations validating our predictions, a number of extra point-to-point links can be added to the torus topology at any point in the simulation.

The network links in the base network are 16 bits wide and are clocked at 100 MHz. In the reported experiments, the characteristics of an elink were assumed to be equal to those in the base network, yielding a per-hop latency that is the same for an elink as for a single base network link. However, our simulation and prediction methodology allow for any other latency ratio. Both coherency traffic (read requests, invalidation messages, etc.) and data (the actual cache lines) are sent over the network. The resulting remote memory access times are representative for a Sun Fire server (around 1 μs on average).

Since the simulated caches are not infinitely large, the network traffic will be the result of both coherency misses and cold/capacity/conflict misses. To make sure that private data transfer does not become excessive, a first-touch memory allocation was used that places data pages of 8 KiB on the node of the processor that first references them.

The SPLASH-2 benchmark suite [17] was chosen as the workload. It consists of a number of scientific and technical applications and is a good representation of the real-world workload of large shared-memory machines. Because the default benchmark sizes are too big to simulate their execution in a reasonable time, smaller problem sizes were used. Since this influences the working set, and thus the cache hit rate, the level 2 cache was resized from the 8 MiB on a real UltraSPARC III to 512 KiB, resulting in an 80% hit rate.

The simulation slowdown (simulated time versus simulation time) was a factor of 50,000 resulting in execution times of several hours per benchmark on a Pentium 4 running at 2.6 GHz with 2 GiB RAM.

4. Predicting memory access speedup

4.1. Overview

In Section 2.2, it was assumed that the network can make n connections between arbitrary node pairs, and that we would choose to connect those node pairs that communicate most intensely. A restriction is made on the node fanout, such that at most f elinks can connect to any one node. Since both the time to compute a new configuration and the switching time are finite,[2] the elinks have to stay in place for some larger period of time, called the reconfiguration interval Δt. After every interval of length Δt, we collect the communication pattern of the last interval, and place the n elinks between the selected node pairs that had the most intense communication. We now derive a prediction of memory access speedup that can be parameterized for n, f and Δt, which are the most important parameters for our reconfigurable network.

The speedup prediction is derived using the following steps:

- A single full simulation is done of each benchmark, using a non-reconfigurable network, yielding a list of memory accesses and a list of network packets.
- Using the list of network packets, the n node pairs that will be connected with an elink are found for each interval.
- The memory accesses that would benefit from the elinks are identified.
- The latency of each memory access is reviewed, for accesses that benefit from an elink this latency is divided by a certain factor.
- A new average memory latency is computed, providing a measure of network performance.

In the rest of this section, each of the above stages is explained in more detail.

4.2. Full simulation

We start by doing one full-system simulation (per benchmark), using the platform described in Section 3. Only the base network is active, so this simulation also serves as the baseline against which we measure the memory latency improvement made by a reconfigurable network. Our simulator creates a list of memory references that cannot be satisfied by the local node, and a second list of all packets that were sent through the network. Each memory reference is annotated with the time the request started, the requesting node, the home node and the measured access latency. For network packets, we store the sending time, the source and destination nodes and the packet size.

[2] In our simulations both are assumed to be zero, but they are reflected in the reconfiguration interval since the sum of selection and switching times should be 'significantly' shorter than the interval.

4.3. Determining the elinks placement

The packet trace is divided into intervals of length Δt. For each interval, sums are made of the number of bytes that were exchanged between each of the $p(p-1)/2$ node pairs (with p the number of processors or nodes). The elinks are bidirectional, so traffic in both directions must be added together. The node pairs are sorted according to the traffic they exchanged in this interval. This list is traversed in descending order, selecting node pairs that will be connected with an elink. As long as we have not found n node pairs, every next node pair on the list is selected unless adding an elink between them violates the fanout restriction.

Instead of blindly using the n links with the most traffic, some optimizations could be made. It is for instance possible that a node pair in the top n is already directly connected by a link from the base network. In this case, the distance between the node pair is already minimal and placing an elink between them will not improve latency, unless adding this link can significantly decrease congestion. This situation is further examined in Section 5.2.2.

Also, in our simulation, elinks are only used for traffic that has the same endpoints as the link, not for traffic that might use the elink as only a part of its path. Solving this limitation would require elink selection and routing protocols that are significantly more complex than those in the current network models, and may not be compatible with the high-speed low-latency environment inside a shared-memory machine. Therefore, we have decided to forego on this issue for now, and delay its study for future work.

4.4. Correlating memory accesses

The metric that makes network performance visible to the processors is the memory access latency. Therefore, we now determine how this memory access latency is affected by the selection of n node pairs. Every memory access that requires network traffic is initiated by the processor on one node and serviced by the directory on another node, the home node of the memory word. We now connect this memory access to the node pair made up by these two nodes. If this node pair was selected for the interval in which the memory access is made, the access is considered to benefit from an elink. Since memory access latencies (around $1\,\mu$s) are significantly shorter than the considered reconfiguration intervals ($100\,\mu$s and upwards), there should be no problems of accesses spanning several intervals.

There are memory accesses that require intervention by a third node, in particular if the memory access is a write and some third node needs to invalidate or write back the word. However, these transactions involving three or more nodes are not very common (in our simulations, their fraction in total memory access latency was always less than 10%). Besides, about half the time of these accesses is still spent in communicating between the two primary nodes, so we pretend these transactions only use the primary nodes.

4.5. Calculating new latencies

Traffic that can use an elink will reach its destination in only one hop, compared to potentially several hops for traffic using the base network. For the 4×4 torus network used in our simulation, the average distance between node pairs is 2.13 network hops. We expect the node pairs selected for connection with an elink to be uniformly distributed over the network, the average distance spanned by an elink should therefore also be 2.13. Traffic between a selected node pair only uses the elink, so the number of hops in this case is reduced to one. This traffic will therefore, on average, traverse a factor of 2.13 times fewer nodes than traffic using the base network. We now assume that memory accesses between selected nodes will have a latency that is reduced by the same factor.[3]

For each memory access we know the source and destination node, and the distance between them. Therefore, we could customize the speedup for each separate access instead of using the average value. However, the placement of subprocesses and data on the nodes may change between different simulation runs, and thus also between the original, base network only simulation and an execution with the reconfigurable network. Therefore, we have decided to use the average latency reduction for all accesses.

The reduction in congestion when using more links, and the fact that wormhole routing is used, both result in the latency not always scaling linearly with the number of hops a packet should traverse. Modeling congestion is, however, not trivially done in the current setting, and has not been attempted for this study.

To summarize: in our original simulation all traffic uses the base network, so the relation between memory access latency in the original simulation with a non-reconfigurable network, and a simulation using a reconfigurable network is the following: memory access latencies between selected node pairs are divided by 2.13, the others are not affected. This relationship remains the same for different values of n, f and Δt, but different sets of memory accesses will benefit from the elinks.

4.6. Estimating application speedup

Total execution time consists of processor *computation time* (for our purpose this includes cached and local memory accesses) and remote memory accesses or *communication time*. The former is in principle not affected by the network architecture, whereas a fraction of the latter (those memory accesses that were identified to benefit from an elink) can be speeded up.

We could try to estimate the application speedup using computation and communication time, and have done so in [13]. However, it became obvious that application

[3]This assumes the base network links and the elinks have the same properties. If this is not the case, the 2.13 ratio can be adjusted accordingly.

performance is not just a function of memory latency, but that other factors introduce variability in the execution time. This makes the application speedup rather unstable, and makes it difficult to compare a simulated speedup with the predicted speedups which are not affected by this variability. Therefore, we will use the improvement of the average memory access latency, as compared to the baseline simulation for the same benchmark, as the performance indicator for the various network implementations.

5. Results and discussion

5.1. Results

Predictions for a number of benchmarks and network parameters are shown in Fig. 5. Each graph shows the results for one benchmark, the final graph averages results across the four benchmarks shown. In each graph, three main groups represent a reconfiguration interval (Δt) of 100 μs, 1 or 10 ms. The fourth group shows the case in which the same number of elinks is added, but they are not reconfigured throughout the length of the program (the best performance out of three random link placements is shown). In the four sub-groups, the number of elinks n is 4, 8, 12 and 16, respectively. The left bar in each sub-group shows the improvement of the average memory access latency as measured by a simulation with the reconfigurable network in place, the right bar shows our prediction of this latency improvement. For all simulations, a fanout restriction of two elinks per node was imposed.

We see that latency generally improves when adding elinks, or when decreasing the reconfiguration interval. This could be expected, since more links allow a larger percentage of memory accesses to be influenced, and a shorter interval allows the network to follow the traffic dynamics more closely. Adding non-reconfigurable elinks (the ‘fixed’ situation in Fig. 5) also improves performance, but less than even the most slowly reconfiguring implementation: just four elinks, reconfiguring every 100 μs, can outperform 16 fixed elinks (on a total of 32 base network links for the 4×4 torus topology, these 16 links correspond to a 50% increase in total network bandwidth).

Somewhat unexpected is that for some benchmarks, in the $\Delta t = 10$ ms case the improvement goes down when adding more links. This is caused by randomness in the simulations, and can be attributed to effects like the behavior of the operating system scheduler moving processes to a different CPU or the convergence in the algorithm of some of the benchmarks. A change in either of these can cause changes in the traffic patterns, and make congestion appear or disappear throughout the network. For instance, the paths between two heavily communicating node pairs may now partly overlap where they previously did not, resulting in added congestion on the network links shared between the paths. This results in a different packet latency and hence a modified memory access latency.

Comparison of the simulated latency improvements with their predicted counterparts shows that our model can provide a reasonably accurate absolute prediction of latency improvement, and, more importantly, a very good relative prediction over the different networks. This allows one to use our model instead of simulations to evaluate trade-offs during the design of a new network architecture. The unexpected behavior in the $\Delta t = 10$ ms case described above does not manifest itself in the predictions. Therefore, our prediction has the added advantage that its results do not suffer from this noise that is present in the full-system simulations.

Fig. 6 shows the computation times required for both the full-system simulations and our prediction model, the former taking several hours while the latter can be completed in just a few minutes. We did not include the cost of the initial simulation in the computation time for our method, since this should only be done once and can subsequently be reused for thousands of network parameter sets. Our method, therefore, allows a reduction in computation time by about two orders of magnitude. Note that we already employed scaled-down benchmarks and simplified architectural models (an in-order processor, less-than-cycle-accurate processor and network models). If one were to do these simulations with a highly detailed simulator the computation time can easily be an order of magnitude higher. In contrast, the prediction model was implemented by a Python script, optimized for maintainability and extensibility. A speed-optimized implementation written in, for instance, C would make the difference in computation time even larger.

5.2. Discussion

5.2.1. Access latency reduction

We have assumed that the average link distance, and therefore also the reduction of this distance after redirecting traffic through an elink, is 2.13. It is conceivable that the operating system, or the algorithms used in the benchmarks, distributes data such that most data are found on a node close by. However, this is not the case: we found the average distance spanned by the elinks to be 2.1 ± 0.1 for all benchmarks. Network traffic using one of the elinks will indeed traverse, on average, 2.13 times fewer hops.

The relationship between reducing the number of hops and the reduction in memory access latency is not very clear at this point. Congestion seems to play an important role, especially since the reduction in congestion (measured as the time a packet spends waiting in a buffer, as opposed to actually moving through the network) is significant (from 50% of total traffic latency in the initial simulation to 15% when elinks are added, averaged across all benchmarks). The effect of a lowered congestion is not included in our model, this results in a systematic over-estimation of the latency (or under-estimation of the latency improvement) as can be seen in Fig. 5. A deeper analysis of the impact of congestion is therefore necessary,

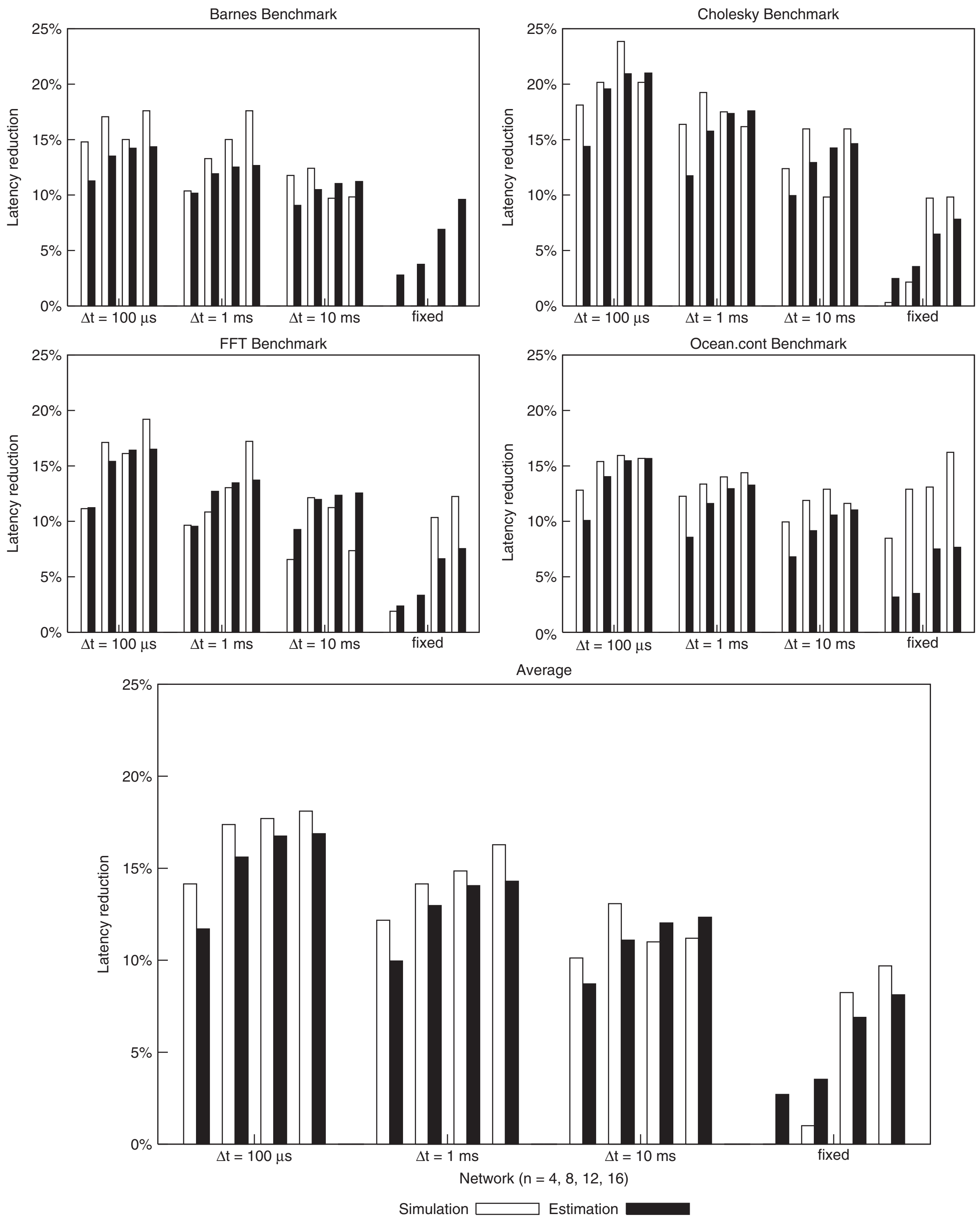


Fig. 5. Measured and predicted average memory access latency reduction on 12 different networks for the Barnes, Cholesky, FFT and Ocean.cont benchmarks (a)–(d), and averaged across all four benchmarks (e). $\Delta t = 100\,\mu s$, 1 ms, 10 ms and ∞ for each of the four outer groups, $n = 4, 8, 12$ and 16 for the inner groups. $f = 2$ for all cases.

including its distribution across the network and its influence on memory accesses, both those on and off the critical path of the application.

Finally, although network latency comprises the largest part of total memory access latency, it is not the only part. Congestion can also occur at the remote memory itself: when a large number of nodes is sending requests to the same node, its network link, the cache controller or the memory bus can become saturated. This is the case for the Cholesky benchmark, where adding elinks does not

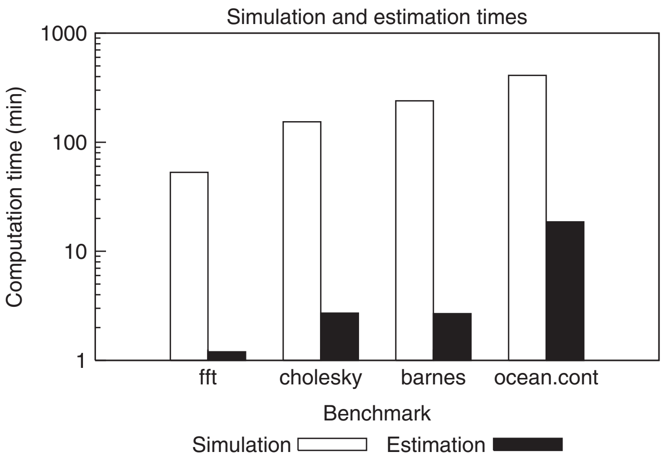


Fig. 6. Computation time (in minutes) required for a full simulation (left) and our prediction (right) for the four benchmarks considered. Note that the *Y*-axis is in a logarithmic scale.

always increase performance (Fig. 5(b), $\Delta t = 1\,\text{ms}$ case). The prediction model does not know about this saturation, and therefore tends to under-estimate latency (or over-estimate the latency improvement).

5.2.2. *Improving elink placement*

When selecting the node pairs to receive a connection with an elink, we have only looked at the traffic that flowed between those nodes, not directly at the benefit such an elink would provide. For nodes that are only one hop apart in the base network, an elink between them will of course not further improve latency.[4] Also, elinks that span a large distance (i.e., the distance between the endpoints of the link, measured over the base network, is large) should provide more gain than elinks that span shorter distances. We have, therefore, modified the elink selection algorithm used in Section 4.3 to sort the node pairs by traffic $\times$ distance instead of sorting just by the traffic they exchanged. This way, a node pair at distance 2 should exchange twice as much traffic as a node pair at distance 4 to have the same chance of being selected for an elink connection. The position of a node pair in the sorted list should now better represent its contribution to the total latency, hopefully making the elinks more effective.

Fig. 7 shows some results for this modified elink selector when simulating the FFT benchmark. The left graph shows the distribution of the distance the elinks span, both for the original selector and for the modified selector. Clearly, the modified selector more often selects node pairs that are far away for connection with an elink. The average spanning distance for all elinks indeed goes up from 2.20 to 2.55. The right graph shows the fraction of memory accesses that is influenced by the elinks, and the fraction of total access latency these accesses represent. A smaller number of accesses is now influenced, however, this smaller fraction represents a slightly larger fraction of the total latency. The latency of the affected memory accesses should also be reduced by a larger factor (since the average elink distance is now larger), so latency improvement should be larger compared to the original selector.

Our predictor can easily be modified to make predictions for this new case. First, we have to change the elink-selection in step 2 of the algorithm detailed in 4.1 so that it mirrors the behavior of our modified selector. Next we have to calculate the reduction factor that operates on the latency of affected memory operations. The average elink distance will no longer be 2.13 since longer links are favored. Since the likelihood of a node pair being selected now grows linearly with the distance between the nodes (assuming equal traffic), the average elink spanning distance will be a weighted average of the distances between every node pair, with the distance itself as the weight. For the 4×4 torus network used in our simulations, this results in an average of 2.5.

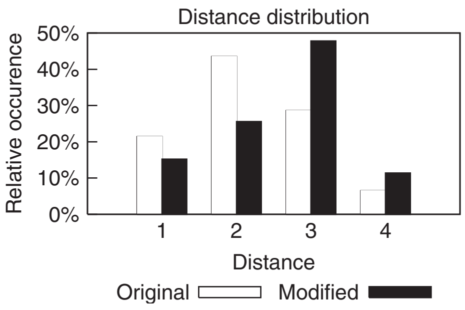
Fig. 8 again shows simulation results and our predictions, for the different networks and averaged across the four benchmarks. Comparing these results with Fig. 5 which employs the original elink selector, we note that the speedup has further improved on all networks. This was to be expected since the elinks are now better distributed to augment the base network.

The absolute prediction accuracy is worse than it was with the original selector. As we did in Section 5.2.1, we again measured the distance spanned by all elinks that were instantiated during the simulations, and found that the average per simulation was now 2.5 ± 0.1, which validates that the average distance an affected packet should travel was now reduced by a factor of 2.5. However, the argument made in 5.2.1 regarding the influence of congestion is probably even more important here and could explain the higher inaccuracy: traffic flows traveling over a large distance cause congestion over a large part of the network; redirecting this traffic over an elink therefore reduces congestion more significantly than redirecting more localized traffic. Since the new selector favors non-localized traffic, the influence of congestion reduction should weigh heavier on the results of Fig. 8, causing a larger systematic over-estimation of the latency. The relative accuracy of the predictor is still high, showing that our method can adapt to different elink selection methods.

6. Future work

As mentioned in Sections 5.2.1 and 5.2.2, the fact that we did not model congestion, and the reduction thereof after adding elinks, causes our prediction method to over-estimate the latency. Further efforts will therefore go into developing an extension to this model that has some notion of the congestion in the network.

[4]It does influence congestion, since direct traffic between the endpoints is now separated from the base network link handling traffic passing through the nodes.



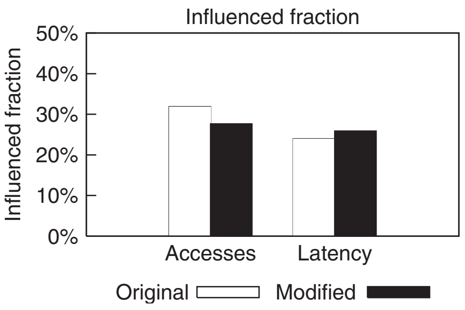


Fig. 7. Spanning distance distribution of the elinks selected by the original and the modified elink selectors (left) and the fraction of memory accesses and of total access latency that is influenced by elinks, also for both selectors (right). All results are for the FFT benchmark with $\Delta t = 1\,\mathrm{ms}$, $n = 8$ and $f = 2$.

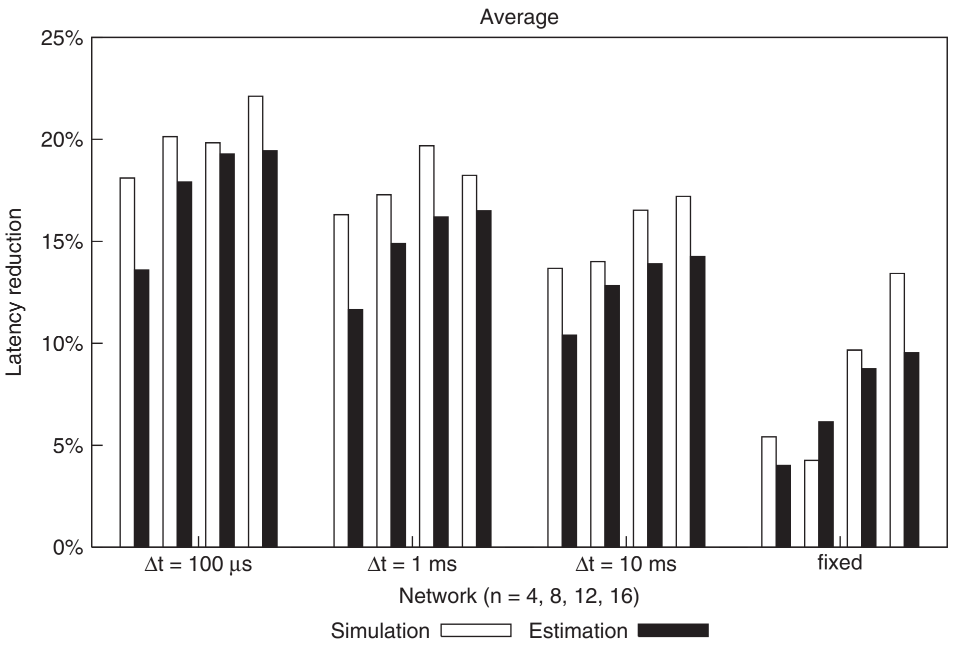


Fig. 8. Measured and predicted average memory access latency reduction on 16 different networks, averaged across all four benchmarks, this time using the modified elink selector. $\Delta t = 100\,\mu\mathrm{s}$, 1 ms, 10 ms and ∞ for each of the four outer groups, $n = 4, 8, 12$ and 16 for the inner groups.

When we look at practical implementations of a reconfigurable network, we note that several of them, including the one presented in [8], do not allow elinks to be placed between arbitrary node pairs. To obtain an efficient use of the elinks in these situations, it is necessary that both elinks and base network links can be in the path of a single packet. This, however, does not allow us to use a single factor (the 2.13 or 2.5 for our 4×4 torus network) to represent the average distance reduction when using an elink. Also we will have to extend routing protocols and develop a more complex elink selection algorithm to handle this case.

7. Conclusions

In this paper, we have addressed the problem of evaluating and designing a partially reconfigurable interconnect network for shared-memory multiprocessors. We have proposed a technique for predicting the average memory access latency for variable network parameters (number of extra links n, the fan-out f, the reconfiguration interval $\Delta t, \ldots$) based on a single simulation run per benchmark and per base network configuration. We found that our prediction has a high relative accuracy and can, therefore, be used to evaluate design trade-offs between different network implementations. The absolute inaccuracy (an almost systematic over-estimation of memory access latency) is probably due to congestion reduction, which is not yet incorporated in our model. Future work will be aimed at including congestion effects in our prediction model. We will also attempt to improve the link selection method and adapt our prediction model accordingly.

Acknowledgments

This paper presents research results of the PHOTON Inter-university Attraction Poles Program (IAP-Phase V), initiated by the Belgian State, Prime Minister's Service, Science Policy Office. C. Debaes is indebted to the FWO for his post-doctoral fellowship.

References

[1] D.A.B. Miller, H.M. Ozaktas, Limit to the bit-rate capacity of electrical interconnects from the aspect ratio of the system architecture, J. Parallel Distributed Comput. 41 (1) (1997) 42–52.

[2] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J.L. Hennessy, M. Horowitz, M.S. Lam, The Stanford DASH multiprocessor, IEEE Comput. 25 (3) (1992) 63–79.

[3] J. Collet, D. Litaize, J.V. Campenhout, M. Desmulliez, C. Jesshope, H. Thienpont, J. Goodman, A. Louri, Architectural approach to the role of optics in monoprocessor and multiprocessor machines, Appl. Opt. 39 (2000) 671–682.

[4] A.F. Benner, M. Ignatowski, J.A. Kash, D.M. Kuchta, M.B. Ritter, Exploitation of optical interconnects in future server architectures, IBM J. Res. Dev. 49 (4/5) (2005) 755–776.

[5] M. Brunfaut, et al., Demonstrating optoelectronic interconnect in a FPGA based prototype system using flip chip mounted 2D arrays of optical components and 2D POF-ribbon arrays as optical pathways, in: Proceedings of SPIE, Bellingham, vol. 4455, 2001, pp. 160–171.

[6] L. Chao, Optical technologies and applications, Intel Technol. J. 8(2).

[7] L. Schares, et al., Terabus—a waveguide-based parallel optical interconnect for Tb/s-class on-board data transfers in computer systems, in: Proceedings of the 31st European Conference on Optical Communication (ECOC 2005), vol. 3, The Institution of Electrical Engineers, Glasgow, Scotland, 2005, pp. 369–372.

[8] I. Artundo, L. Desmet, W. Heirman, C. Debaes, J. Dambre, J. Van Campenhout, H. Thienpont, Selective optical broadcast component for reconfigurable multiprocessor interconnects, IEEE J. Sel. Top. Quantum Electron., Special issue on Optical Communication, 2006, in print.

[9] C. Katsinis, Performance analysis of the simultaneous optical multi-processor exchange bus, Parallel Comput. 27 (8) (2001) 1079–1115.

[10] W. Heirman, J. Dambre, J. Van Campenhout, C. Debaes, H. Thienpont, Traffic temporal analysis for reconfigurable interconnects in shared-memory systems, in: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, Colorado, IEEE Computer Society, Denver, Colorado, 2005, p. 150.

[11] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, Simics: a full system simulation platform, IEEE Comput. 35 (2) (2002) 50–58.

[12] F. Ridruejo, A. Gonzalez, J. Miguel-Alonso, TrGen: a traffic generation system for interconnection network simulators, in: First International Workshop on Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems (PEN-PCGCS'05), Olso, Norway, 2005.

[13] W. Heirman, J. Dambre, D. Stroobandt, J. Van Campenhout, C. Debaes, H. Thienpont, Prediction model for evaluation of reconfigurable interconnects in distributed shared-memory systems, in: Proceedings of the 2005 International Workshop on System Level Interconnect Prediction (SLIP'05), ACM Press, San Francisco, CA, 2005, pp. 51–58.

[14] T. Sterling, D. Savarese, D.J. Becker, J.E. Dorband, U.A. Ranawake, C.V. Packer, Beowulf: a parallel workstation for scientific computation, in: Proceedings of the International Conference on Parallel Processing, CRC Press, Boca Raton, USA, 1995, pp. 11–14.

[15] T.M. Pinkston, J.W. Goodman, Design of an optical reconfigurable shared-bus-hypercube interconnect, Appl. Opt. 33 (8) (1994) 1434–1443.

[16] J.L. Sánchez, J. Duato, J.M. García, Using channel pipelining in reconfigurable interconnection networks, in: Sixth Euromicro Workshop on Parallel and Distributed Processing, 1998.

[17] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, A. Gupta, The SPLASH-2 programs: characterization and methodological considerations, in: Proceedings of the 22nd International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, 1995, pp. 24–36.

Wim Heirman was born in Temse, Belgium, on November 28, 1980. He received the M.Sc. degree in computer engineering from Ghent University, Ghent, Belgium, in 2003. He is currently doing Ph.D. research with the Department of Electronics and Information Systems (ELIS), Ghent University. His current research interests include parallel computing systems, reconfigurable architectures and interconnection networks.

Joni Dambre was born in Ghent, Belgium, in 1973. She received the M.Sc. degree in electrotechnical engineering, and the Ph.D. degree in computer engineering from Ghent University, Ghent, Belgium, in 1996 and 2003, respectively. She is currently a Postdoctoral Researcher with the Department of Electronics and Information Systems (ELIS), Ghent University. Her research interests include early evaluation of new interconnect techniques in digital systems. Dr. Dambre is a Member of ACM.

Iñigo Artundo was born in Pamplona, Spain on October 21, 1979. In 2004, he received with the greatest distinction his master degree in telecommunication engineering at the Public University of Navarra. Currently, he is doing a Ph.D in the field of reconfigurable optical interconnects architectures at the Department of Applied Physics and Photonics, Vrije Universiteit Brussel, Belgium. His current research interests are reconfigurable architectures, optical interconnection networks and distributed shared-memory systems.

Christof Debaes was born in Geraardsbergen, Belgium, in 1975. He graduated as an electrotechnical engineer from the Vrije Universiteit Brussel (VUB) in 1998. He received the Ph.D. degree from the Applied Physics and Photonics Department, VUB, in collaboration with the Ginzton Laboratory, Stanford University, Stanford, CA, directed by Prof. D.A.B. Miller. He is currently working at the VUB on a postdoctoral fellowship from the Flemish Fund for Scientific Research (FWO-Vlaanderen). His research activities are focused on optical interconnects covering a wide range of subjects such as optical clock injection, opportunities for reconfigurable optical interconnect and the use of the use Deep Proton Lithography for micro-optical components.

Hugo Thienpont was born in Belgium 1961. He graduated from the Vrije Universiteit Brussels (VUB) in 1984 as an Electrical Engineer with majors in applied physics and applied optics. In 1994, he became Professor in the Faculty of Applied Sciences. Today he is director of research of the "Laboratory for Photonics" and is promoter of different photonics-related research and industrial projects. His research activities comprise materials, modeling, components and devices, packaging and demonstrators for photonic interconnects.

Dirk Stroobandt obtained the Ph.D. degree in electrotechnical engineering in 1998 from Ghent University. From 1998 Dirk Stroobandt was Post-doctoral Fellow with the Fund for Scientific Research-Flanders (Belgium) (F.W.O.). Since October 2002, he is full professor at Ghent University where he is affiliated with the Department of Electronics and Information Systems (ELIS). His research is oriented towards a priori estimations of interconnection lengths in electronic systems and hardware/software codesign for embedded systems.

Jan Van Campenhout was born in Vilvoorde, Belgium, on August 9, 1949. He received the degree in electromechanical engineering from Ghent University, Ghent, Belgium, in 1972; and the M.S.E.E. and Ph.D. degrees from Stanford University, Stanford, CA, in 1975 and 1978, respectively. He is currently with the Faculty of Engineering, where he teaches courses in computer architecture, electronics, and digital design, and is also the Head of the ELIS Department, at Ghent University. His current research interests include the study and implementation of various forms of parallelism in computer systems, and their application in programming language support, computer graphics and robotics. He is a Member of Sigma Xi, KVIV, and ACM.