

# Predicting the Performance of Reconfigurable Interconnects in Distributed Shared-Memory Systems

W. Heirman, J. Dambre, J. Van Campenhout  
Universiteit Gent, ELIS  
Sint-Pietersnieuwstraat 41  
9000 Gent, Belgium  
wheirman@elis.ugent.be

*Abstract*— New advances in reconfigurable optical interconnect technologies will allow the fabrication of low-cost, fast and run-time adaptable networks for connecting processors and memory modules in large distributed shared-memory multiprocessor machines. Since the switching times of these components are typically high compared to the memory access time, reconfiguration exploits low frequency dynamics in the network traffic patterns. These are however not easily reproduced using statistical traffic generation, the tool of choice when doing a fast design space exploration. In this paper, we present a technique that can accurately predict network performance, based on the traffic patterns obtained from simulating the execution of real benchmark applications, but without the need to perform these slow full-system simulations for every parameter set of interest.

*Keywords*— prediction, reconfiguration, interconnect, shared-memory

## I. INTRODUCTION

The electrical interconnection networks connecting the different processors and memory modules in a modern large-scale multiprocessor machine, are running into several physical limitations [11]. In shared-memory machines, where the network is part of the memory hierarchy [9], 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]. Mostly unhindered by crosstalk, attenuation and capacitive effects, 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 [1], [2] 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 [4], [7] that supports a much higher bandwidth than that allowed by 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 patterns. We have already characterized the locality in both time and space of the traffic flowing over the network [6], using large-scale simulation of the execution of real benchmark programs with a simulation platform based on the Simics multiprocessor simulator [10]. We have found that long periods of intense communication occur between some node pairs suggesting that slowly reconfiguring networks can result in a significant application speedup. Next, we included the model of a specific reconfigurable network in our simulator, and measured the attained application speedup [4].

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 requires tens or even hundreds 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. Hence, one such simulation can take several hours to com-

plete, so it is very unpractical to do a full-system simulation for each benchmark application and each set of network parameters. The typical solution for this problem is to do away with the full-system simulation, and only model the interconnection network. The network traffic is now no longer generated by an actual parallel application, but by a statistical traffic generator [13]. These traffic generators are usually good for modeling simple traffic such as uniform distributions or broadcasting behavior, which can suffice to evaluate static networks. Our reconfigurable networks however depend on low-frequency dynamics of the network traffic such as bursts, which are not sufficiently modeled in existing traffic generators. This precludes their use for our purposes, leaving us with the much slower full-system simulations.

A step between full-system simulation, also referred to as execution-driven because the network traffic is generated by the simulated execution of a parallel program, and statistical traffic generation is trace-based simulation. Here, one full-system simulation is performed and the resulting traffic flow is recorded. The flow is later *played back* and fed into the simulator of an interconnection network with different parameters. Since the network traffic usually does not depend too much on the timing of the specific interconnection network, the traffic that is used is very similar to the traffic that would have been generated using an execution-driven simulation with the new network, resulting in an acceptable approximation. At the same time, the removal of the processors and caches from the simulation results in a greatly reduced simulation time.

The method we present in this paper is based on trace-based simulation, but goes one step further: we start with a trace of the traffic from one full-system simulation, but we do not actually simulate the flow of packets in our subsequent evaluation of the different networks. Since the reconfiguration of the network exhibits itself only as a modification of the network topology, it is very easy to determine the distance a packet will have to travel in the new network as compared to the old network, allowing us to rapidly predict new packet latencies, and estimate, to a certain level of accuracy, the resulting network performance. This way we can very rapidly explore the design space of the interconnection network and make a (albeit crude) comparison between different proposed network architectures.

This paper reports on our prediction methodology and analyzes the results. Section II describes in more

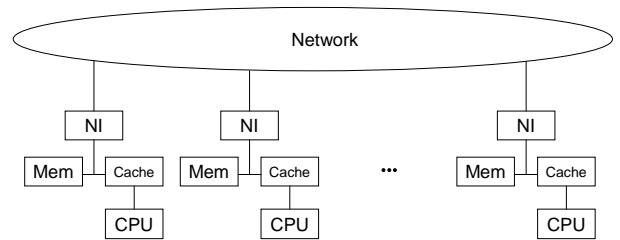


Fig. 1. Schematic overview of a multiprocessor machine. For message-passing machines, the network traffic is under control of the application. In shared-memory machines, network traffic is generated by the network interfaces (NI) in response to non-local memory accesses by a processor.

detail the architecture of both the shared-memory machine and the reconfigurable network that were used in this study. In section III, the methodology that was followed to obtain the communication patterns is described. The prediction method is presented in section IV. Section V gives the prediction results and compares them with the actual speedup. In section VI, some future work is discussed, the conclusions are summarized in section VII.

## II. SYSTEM ARCHITECTURE

### A. 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 to each other (figure 1). In the extreme end of the loosely coupled family we find examples such as the *Beowulf cluster* [15], in which the network consists of a commodity technology such as Ethernet. This simplistic interconnection network results in relatively low throughput (1 Gbps per processor) and high latency (up to several ms, for a large part due to protocol overhead). 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, 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 hierarchy, it also makes such machines much more vulnerable to increased network latencies.

Modern examples of this class of machines range from small, 2- or 4-way SMP server machines, 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 frequencies, they make very likely candidates for the application of reconfigurable optical interconnection networks.

For this study we consider a directory based coherence protocol, which was pioneered in the Stanford DASH multiprocessor [9]. 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 which 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 coherence 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 consist of both coherence-related traffic (control packets such as invalidate requests) and data traffic (words that were not in a cache due to cold, conflict, capacity or coherence 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. This makes it very difficult to effectively hide the communication latency, and makes the system performance very much dependent on network latency.

### B. A simple reconfigurable network architecture

Previous studies concerning reconfigurable networks have mainly dealt with fixed topologies (usually

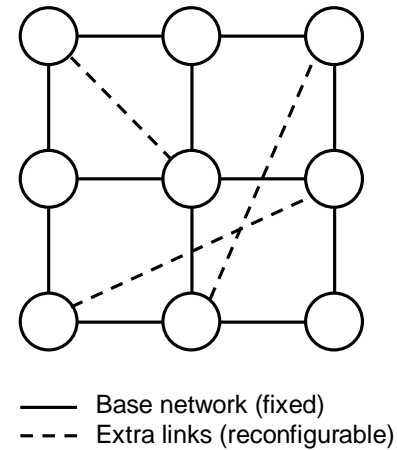


Fig. 2. Reconfigurable network topology. The network consists of a base network, augmented with a limited number of direct, reconfigurable links.

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 [12], [14]. 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* for short. A schematic overview is given in figure 2. An advantage of this setup, compared to other topologies that allow for more general reconfiguration, is that the base network is always available, which is most important during periods where the extra network is undergoing reconfiguration and may not be usable. Routing and reconfiguration decisions are also simplified because it is not possible to completely disconnect a node from the others – the connection through the base network will always be available.

Reconfiguration takes place at specific intervals, the length of each interval being a (fixed) parameter of the network architecture. Traffic is observed by the 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 extra links are determined, within the constraints of the network architecture, such that node pairs that exchanged the most data in the previous interval will be ‘closer together’: the distance, defined as the number

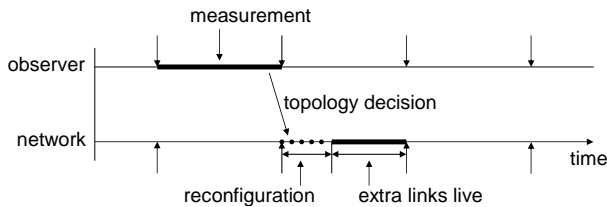


Fig. 3. The observer measures network traffic, and at certain intervals makes a decision where to place the extra links. Reconfiguration will then take place, during which time the extra links are unusable.

of hops a packet sent between the pair must traverse in the new network topology, is minimized. This way a large percentage of the traffic has a short path and a correspondingly low uncongested latency, also congestion is lowered because heavy traffic is no longer spread out over a large number of links.

After selecting the new network configuration, the network is reconfigured and a new interval begins (figure 3). For now, we assume that both the computation to select the new elinks and the physical reconfiguration (done by switching mirrors, tuning lasers etc.) are performed instantaneous. This is not the case in reality (depending on the technology, reconfiguration can take from tens of microseconds up to several milliseconds), but as long as the reconfiguration interval is chosen large compared to the computation and switching times, this is a good enough approximation. On the other hand, if the reconfiguration interval is too long, the elink placement for one interval, based on traffic measurements from the *previous* interval, will not be a good match for traffic in the current interval, degrading the performance improvement obtained.

### III. METHODOLOGY

#### A. Simulation platform

We have based our simulation platform on the commercially available Simics simulator [10]. It was configured to simulate a multiprocessor machine based on the Sun Fire 6800 Server, with 16 UltraSPARC III processors at 1 GHz running the Solaris 9 operating system. Stall times for caches and main memory are set to realistic values (2 cycles access time for L1 caches, 19 cycles for L2 and 100 cycles for SDRAM). The interconnection network is a custom extension to Simics, and models a 4x4 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 extra (optical) link were assumed to be equal to those in the base network, yielding a per-hop latency that is the same for an extra link as for a single base network link. However, our simulation and prediction methodology allow for any other latency ratio. Both coherence traffic (resulting from the directory based MSI-protocol) and data (the actual cache lines) are sent over the network, and result in memory access times representative for a Sun Fire server (a few hundred nanoseconds on average for accesses that require use of the network). Source, destination and size of each network packet are saved to a log file for later analysis.

To avoid deadlocks, dimension routing is used on the base network. Each packet can go through one extra link on its path, after that it switches to another virtual channel (VC)<sup>1</sup> to avoid deadlocks of packets across extra links. For routing packets through the elinks we use a static routing table: when reconfiguring the network, the routing table in each node is updated such that for each destination it tells the node to route packets either through an elink starting at that node, to the start of an elink on another node, or straight to its destination, the latter two using normal dimension routing.

Since the simulated caches are not infinitely large, the network traffic will be the result of both coherence 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. Also each thread is pinned down to one processor (using the Solaris `processor_bind()` system call), so the thread stays on the same node as its data for the duration of the program.

The SPLASH-2 benchmark suite [16] 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 an actual 8 MiB on a real UltraSPARC III to 512 KiB, resulting in a realistic

<sup>1</sup>Actually another *set* of VCs is used since we already employ separate request and reply VCs to avoid *fetch deadlock* [8] at the protocol level.

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.

### B. Network architecture

To avoid pinning our discussion down on the peculiarities of a specific network architecture, we test our model with a hypothetical parameterized architecture that provides the infrastructure to potentially place an elink between any two given nodes. Two constraints are made on the set of elinks that are active at the same time:

- A maximum of  $n$  extra links can be active concurrently.
- The fanout of each node is limited to  $f$ , not including connections to the base network.

The reconfiguration interval  $\Delta t$  is the third parameter, all results in this paper will be based on different sets of values for these three parameters.

### C. Extra link selection

As explained in section II-B, we want to minimize the number of hops for most of the traffic. Formally, our cost function can be expressed as

$$C = \sum_{i,j} d(i,j) \cdot T(i,j)$$

with  $d(i,j)$  the distance between nodes  $i$  and  $j$ , which is a function of the elinks that are selected to be active, and  $T(i,j)$  the number of bytes exchanged between the node pair in the time interval of interest.

Since the reconfiguration interval will typically be in the order of milliseconds, and making the extra link selection should be done in only a fraction of this time, we need a fast heuristic that can quickly find a set of active elinks that satisfies the constraints imposed by the architecture and has an associated cost close to the global optimum. We have constructed a greedy algorithm that works as follows:

1. A list is constructed of all node pairs  $(i,j)$ , sorted by  $d(i,j) \cdot T(i,j)$  in descending order. Only the base network connections are considered for determining  $d(i,j)$ .
2. Initialize the set of active elinks  $E_a$  to be empty, and the set of possibly active elinks  $E_p$  to contain all elinks that can be supported by the architecture (but not necessarily at the same time). For our test architecture,  $E_p$  would contain all  $p(p-1)/2$  node pairs (with  $p$  the number of processors or nodes).

3. For the node pair at the top of the list, determine which new elink (one that is not already in  $E_a$  but is still in  $E_p$ ) is the *most interesting*, i.e. when enabled, would give the greatest reduction in distance between the nodes. This elink is removed from  $E_p$  and added to  $E_a$ , also, the current node pair is removed from the top of the list. If an elink resulting in a direct connection between the node pair is still available in  $E_p$  this one will of course be selected, if none of the elinks in  $E_p$  can provide a distance lower than the one over the base network or over an elink already in  $E_a$ , no new elink is activated. To quickly do this selection, we precompute a table that gives the distance between each node pair as a function of the activated elinks. Since only one elink is used in each path, this table has a maximum of (number of node pairs)  $\times$  (number of elinks) entries, and usually much less since only a small number of elinks can decrease the base distance for a specific node pair.

4. Once a link has been added to  $E_a$ , we check the constraints imposed by the network architecture. If activation of one of the links in  $E_p$  would cause a node to exceed its fanout limit  $f$ , this elink is removed from  $E_p$  and is therefore no longer considered for activation in the next interval. When the maximum number of elinks  $n$  is reached, the algorithm terminates.

5. As long as there are nodepairs on the list, and the set of possible elinks  $E_p$  is not empty, continue with step 3. Else, end the algorithm,  $E_a$  is now the set of elinks that should be enabled.

Note that, after enabling one of the elinks, one could recompute the distances between node pairs and update the list of node pairs before starting a new iteration at step 3. This is however considered too time-consuming, and has not been implemented in our algorithm.

Using a branch and bound method, it proved possible to determine the elink placement giving the global minimum of  $C$ . This takes however about 1 minute to execute for each given traffic pattern, which underlines the need for a fast heuristic. By comparing the resulting cost of the greedy algorithm with the global minimum, we can confirm the quality of our heuristic since it always resulted in a cost that was at most 10% above the global optimum.

## IV. PREDICTING APPLICATION SPEEDUP

### A. Overview

We will now present our performance prediction for reconfigurable networks, based on only one full-system

simulation run per benchmark. This prediction is parameterized on the values of  $n$ ,  $f$  and  $\Delta t$ , and can therefore predict the performance of a range of candidate networks, while still relying on only a small number of long simulations. For each benchmark, this prediction is derived using the following steps:

- A single full simulation is done of the benchmark, using a non-reconfigurable network (also referred to as the baseline simulation), yielding a list of memory accesses and a list of network packets.
- Using the list of network packets, the traffic exchanged between each node pair is calculated for each interval of duration  $\Delta t$ .
- The placement of the extra links, given the traffic pattern just computed, is determined for each interval.
- The latency of each memory access is reviewed, for accesses that would benefit from an extra link this latency is reduced.
- Using the latency distribution over the different processors, an average speedup and a best and worst case are derived.

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

### B. Full simulation

We start by doing one full-system simulation (per benchmark), as described in section III. Only the base network is active, so this simulation also serves as the baseline against which we calculate the speedup to determine the efficiency of 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 node, the destination node and the packet size. Note that there are only two sizes of messages generated by the coherence controllers: 16 byte packets with only control information (read request, invalidate, ...) and 80 byte packets that contain control information plus a complete cache line of 64 bytes.

### C. Determining the extra link placements

The packet trace is divided into intervals of duration  $\Delta t$ . 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 extra links are bidirectional, so traffic

in both directions must be added together<sup>2</sup>. When we have the traffic profile for the interval, we use the greedy algorithm described in III-C to determine extra link placements.

### D. Correlating memory accesses

The metric that makes network performance visible to the processors, is the memory access latency. It is also the metric that would most easily allow us to say something about application speedup. Therefore we now correlate the memory access latency to the selected elinks. 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 therefore connect this memory access to the node pair made up by these two nodes. We measure the distance between these nodes, both before adding the extra links and after, and tag the memory access with these distances, hereafter called the *baseline distance* and the *elink distance*.

Memory access latencies (at most a few  $\mu s$ ) are significantly shorter than the considered reconfiguration intervals (100  $\mu s$  and upwards), so 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 3 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.

### E. Calculating new latencies

First we calculate the average memory access latency, over the course of the simulation, for all memory accesses with the same baseline distance. This gives us an estimated memory access latency as a function of the distance between its primary nodes. For now, we assume latency is *only* a function of this distance, and does not change by adding extra links.

<sup>2</sup>An optical interconnection (light source  $\rightarrow$  waveguide  $\rightarrow$  detector) is unidirectional, so a *link* consists of two such assemblies. In theory it is therefore not necessary for the extra links 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.

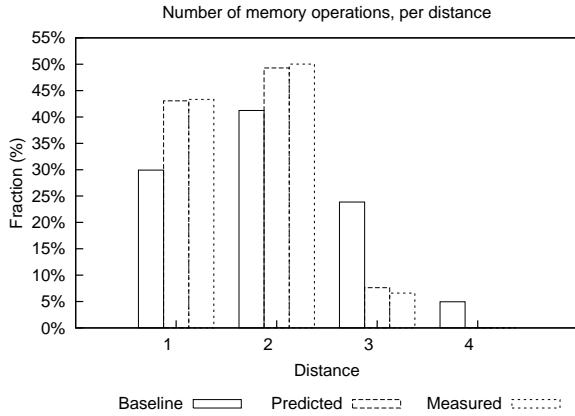


Fig. 4. Baseline, estimated and actual distance distribution of memory operations for the FFT benchmark. The parameters for the reconfigurable network are  $n = 8$ ,  $f = 2$  and  $\Delta t = 1\text{ms}$ .

The old average access latency can be computed as a weighted average of these per distance latencies, with the number of accesses per baseline distance as the weights. We can also count the number of accesses per elink distance and compute a new average using the number of accesses per elink distance as weights, this is our estimate for the average memory access latency after adding the elinks.

Figure 4 shows that we can accurately estimate the number of accesses per elink distance, using the traffic pattern from a simulation with a non-reconfigurable network. For each distance, the graph shows the number of memory accesses in the baseline simulation, the predicted number of accesses using the method described above, and the actual number of accesses, measured in a simulation where the reconfigurable network is added to the machine. We can clearly see that adding a reconfigurable network greatly reduces the average distance, also this new distance can be estimated accurately based on traffic patterns obtained from a baseline simulation run.

The next question is whether memory latency is indeed only a function of distance, and does not vary with topological parameters. Figure 5 shows this memory latency for a few different networks, as measured during simulations with the reconfigurable network in place. For  $d = 4$  the difference is obvious, however,  $d = 4$  accesses are almost eliminated after adding elinks so the error here does not influence the global average. Other distances show much less variation, the difference being due to the reduction of congestion after adding more links to the network.

Figure 6 finally shows the memory access latency

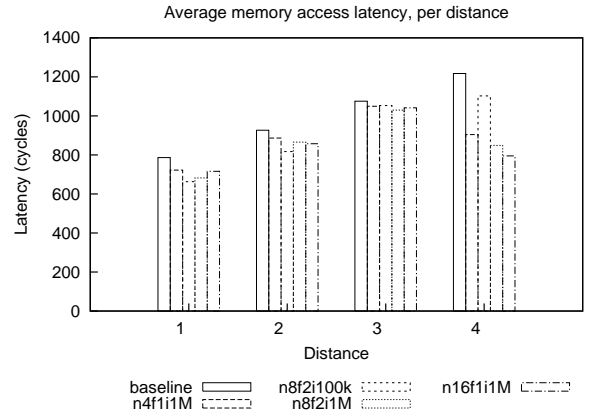


Fig. 5. Variation of average memory latency per distance for different network parameters.

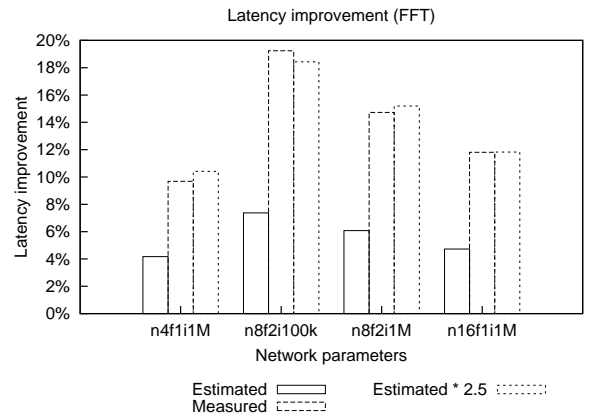


Fig. 6. Latency improvement after adding elinks: estimated, measured, and estimated  $\times 2.5$

improvement over the baseline after adding the extra links. The first columns are our estimates using the method described, the second columns show the measured values in simulation, both for a number of different network parameters. Since we have not included congestion in our method the results are not very good, however, the *relative* prediction accuracy between different network parameters, which is actually the most important value when comparing different suggested network implementations, is rather good. To show this, we added the third columns which are a rescaled version of the first ones by a factor of 2.5. This shows that, although our method does not allow one to accurately predict the performance of one specific network, it can be useful to very quickly compare different network parameters and do an early design-space exploration.

### F. Estimating application speedup

Total execution time consists of processor computation time (including cached and local memory accesses) and remote memory accesses or *communication time*. The former is in principle not affected by the network architecture, whereas the latter can be reduced by modifying the network.

We could consider the minimal reduction in execution time across all processors, assuming that the others will have to wait for the slowest one. It is, however, also possible that the processor with the smallest speedup was actually waiting some part of the time for other processors (a processor spin-locking on a cached synchronization variable does not stall on a remote memory access and would therefore be computing according to our definition), in this case the global speedup will be closer to the *best* speedup across all processors. The identification of the critical processor, determining the actual speedup, is very difficult and highly depending on the specific algorithm and implementation of each benchmark. Therefore, we propose to use the *average* speedup across all processors as our prediction of total speedup, and use the best and worst speedups to denote a confidence interval.

Note that in-order processors are simulated, whereas a modern out-of-order processor can overlap some of the memory access latency with useful computations. However, there is usually a significant dependency between instructions, which makes that even a very aggressive out-of-order processor can only execute a few dozen instructions (corresponding to an equal number of nanoseconds) before blocking on the memory access (which takes something in the order of 1  $\mu$ s). Therefore out-of-order execution should not influence the results too much.

## V. RESULTS AND DISCUSSION

### A. Results

The predictions and confidence intervals for a number of network parameters are shown in figures 7 and 8, for the FFT and `ocean.cont` benchmarks respectively. For each network, the predicted speedup is shown as the first bar. Best and worst estimates are also shown as error bars. The second bar is the result of an actual simulation with the reconfigurable network in place. Again, we have multiplied our estimated memory latency by the empirical factor of 2.5 to show the quality of the *relative* predictions, as opposed to the much more inaccurate *absolute* predictions.

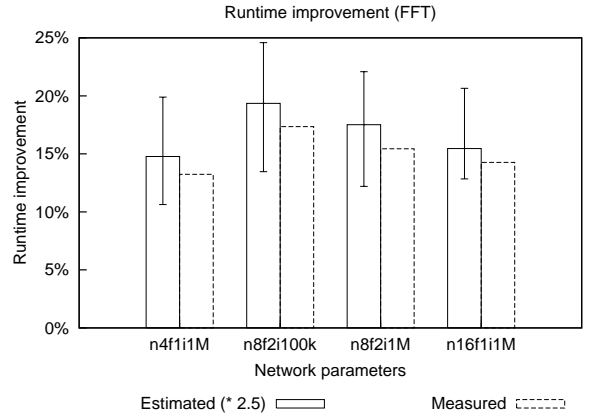


Fig. 7. Runtime improvement for FFT, estimated and measured.

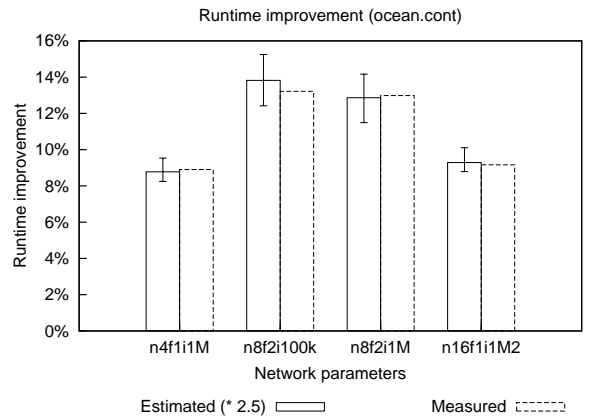


Fig. 8. Runtime improvement for `ocean.cont`, estimated and measured.

### B. Improving accuracy

Most steps of the algorithm described in IV introduce very little error in the final prediction. Figure 4 for instance shows that we can very accurately predict the distance distribution of memory operations after adding a reconfigurable network. The average memory latency is however, as figure 5 clearly shows, not just a function of hop distance, as was assumed in IV-E.

It was attempted to recompute average memory latencies in each interval, mitigating any effects that would make the memory latency dependent on time-varying application behavior. This however did not significantly change the resulting latency, which shows that average per distance memory latency is not a function of time.

Application variability, or the fact that an application might take a different control path when network timings change, can also reduce the accuracy of our runtime predictions for some benchmark applications.

We have analyzed the influence of this effect in greater detail in [5]. Some of the error of the runtime predictions in figures 7 and 8 is caused by this effect, to an extent dependent on the benchmark application. However, this effect does not influence the predictions of the average memory latency (figures 4 through 6) and is therefore not responsible for the empirical value 2.5 introduced there.

The remaining component with the largest influence is probably the reduction of congestion. Clearly, adding links to the network increases bisection bandwidth and generally increases the capacity of the network. We even place our elinks such that large traffic flows are moved away from the base network, speeding up not only the traffic that was moved but also the traffic that remains on the base network. Further analysis of the impact of congestion is therefore necessary.

### C. Reduction in simulation time

The execution of our prediction method, for one benchmark and one set of network parameters, takes just a few minutes. Without our method a full-system simulation would be needed, requiring several hours of processor time. The evaluation of a new network can therefore be performed about 50 times faster, not including the initial cost of the baseline simulation.

## VI. FUTURE WORK

Several assumptions were made in the prediction model, they have been stated throughout section IV. Some of them are fairly trivial, others need more work to either validate or invalidate them. Most importantly, the inclusion of congestion effects into our model should be our next goal.

## VII. CONCLUSIONS

In this paper, we have addressed the problem of evaluating and designing a partially reconfigurable interconnection network for shared-memory multiprocessors. We have proposed a technique for predicting the average network latency and total runtime for variable network parameters (number of extra links  $n$ , node fan-out  $f$ , reconfiguration time  $\Delta t$ , link latency, ...) based on a single simulation run per benchmark. We have also defined confidence intervals for our prediction and found that relative performance over different network parameters can be accurately predicted, but that our method is currently insufficient to provide absolute performance numbers. We have analyzed the impact of several assumptions made

in our model, and found the omission of congestion modeling to be responsible for the largest errors. Future work will be aimed at modeling these congestion effects and incorporating them in our prediction model.

## VIII. 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.

## REFERENCES

- [1] 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*, vol. 4455, Bellingham, July 2001, pp. 160–171.
- [2] L. Chao, "Optical technologies and applications," *Intel Technology Journal*, vol. 8, no. 2, May 2004.
- [3] J. Collet, D. Litaize, J. V. Campenhout, M. Desmulliez, C. Jesshope, H. Thienpont, J. Goodman, and A. Louri, "Architectural approach to the role of optics in monoprocessor and multiprocessor machines," *Applied Optics*, vol. 39, pp. 671–682, 2000. [Online]. Available: <http://www.laas.fr/collet/A02000.pdf>
- [4] W. Heirman, I. Artundo, D. Carvajal, L. Desmet, J. Dambre, C. Debaes, H. Thienpont, and J. Van Campenhout, "Wavelength tuneable reconfigurable optical interconnection network for shared-memory machines," in *Proceedings of the 31st European Conference on Optical Communication*, Glasgow, Scotland, September 2005, *To appear*.
- [5] W. Heirman, J. Dambre, D. Stroobandt, J. Van Campenhout, C. Debaes, and 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*. San Francisco, California: ACM Press, April 2005, pp. 51–58.
- [6] W. Heirman, J. Dambre, J. Van Campenhout, C. Debaes, and H. Thienpont, "Traffic temporal analysis for reconfigurable interconnects in shared-memory systems," in *Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium*. Denver, Colorado: IEEE Computer Society, April 2005, p. 150.
- [7] C. Katsinis, "Performance analysis of the simultaneous optical multi-processor exchange bus," *Parallel Computing*, vol. 27, no. 8, pp. 1079–1115, 2001.
- [8] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, W. D. Hillis, B. C. Kuszmaul, M. A. S. Pierre, D. S. Wells, M. C. Wong-Chan, S.-W. Yang, and R. Zak, "The network architecture of the Connection Machine CM-5," *Journal of Parallel and Distributed Computing*, vol. 33, no. 2, pp. 145–158, 1996. [Online]. Available: <http://citeseer.nj.nec.com/leiserson94network.html>
- [9] D. Lenoski, J. Laudon, K. Gharachorloo, W.-D. Weber, A. Gupta, J. L. Hennessy, M. Horowitz, and M. S. Lam,

- “The Stanford DASH multiprocessor,” *IEEE Computer*, vol. 25, no. 3, pp. 63–79, March 1992.
- [10] P. S. Magnusson, 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*, vol. 35, no. 2, pp. 50–58, February 2002.
- [11] D. A. B. Miller and H. M. Ozaktas, “Limit to the bit-rate capacity of electrical interconnects from the aspect ratio of the system architecture,” *Journal of Parallel and Distributed Computing*, vol. 41, no. 1, pp. 42–52, 1997.
- [12] T. M. Pinkston and J. W. Goodman, “Design of an optical reconfigurable shared-bus-hypercube interconnect,” *Applied Optics*, vol. 33, no. 8, pp. 1434–1443, 1994.
- [13] F. Ridruejo, A. Gonzalez, and J. Miguel-Alonso, “TrGen: a traffic generation system for interconnection network simulators,” in *1st. Int. Workshop on Performance Evaluation of Networks for Parallel, Cluster and Grid Computing Systems (PEN-PCGCS’05)*, Oslo, Norway, June 2005.
- [14] J. L. Sánchez, J. Duato, and J. M. García, “Using channel pipelining in reconfigurable interconnection networks,” in *6th Euromicro Workshop on Parallel and Distributed Processing*, January 1998.
- [15] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer, “Beowulf : A parallel workstation for scientific computation,” in *Proceedings of the International Conference on Parallel Processing, Boca Raton, USA*. CRC Press, August 1995, pp. 11–14.
- [16] 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 22th International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, 1995, pp. 24–36. [Online]. Available: <http://citeseer.nj.nec.com/woo95splash.html>