Prediction Model for Evaluation of Reconfigurable Interconnects in Distributed Shared-Memory Systems

W. Heirman
Ghent University, ELIS
Sint-Pietersnieuwstraat 41
9000 Ghent, Belgium
wheirman@elis.ugent.be

J. Dambre
Ghent University, ELIS
Sint-Pietersnieuwstraat 41
9000 Ghent, Belgium
jdambre@elis.ugent.be

D. Stroobandt
Ghent University, ELIS
Sint-Pietersnieuwstraat 41
9000 Ghent, Belgium
dstr@elis.ugent.be

C. Debaes
Free University of Brussels
Pleinlaan 2
1050 Brussels, Belgium
christof.debaes@vub.ac.be

H. Thienpont
Free University of Brussels
Pleinlaan 2
1050 Brussels, Belgium
hthienpo@vub.ac.be

J. Van Campenhout
Ghent University, ELIS
Sint-Pietersnieuwstraat 41
9000 Ghent, Belgium
jvc@elis.ugent.be

ABSTRACT

Reconfigurable interconnection networks for distributed shared memory machines exploit properties of the workload dynamics that are not easily captured by statistical traffic models. Therefore, when designing such a network, one should make trade-offs based on full-system simulation for all viable workloads. It is however very time-consuming to do such simulations. In this paper, we present a technique that can predict the performance of a machine for different network parameters, based on the results of only one full simulation run. We also define confidence intervals for our prediction, and analyze the impact of several assumptions that were made.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures—*Distributed architectures*

General Terms

performance, design

Keywords

Prediction model, interconnection network, reconfiguration, distributed shared-memory

1. 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 [9]. In shared-memory machines, where the network is part of the memory hierarchy [7], 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 [4]. 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, 3] 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 [6, 2] 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 with high communication between them have a high-bandwidth, low-latency connection.

However, the switching time for some of these components is such that reconfiguration will necessarily have to take place on a time scale that is significantly above that of individual memory accesses. The efficiency with which such networks can be deployed strongly depends 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 [5], using large-scale simulation of the execution of real benchmark programs with a simulation platform based on the Simics multiprocessor simulator [8]. 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.

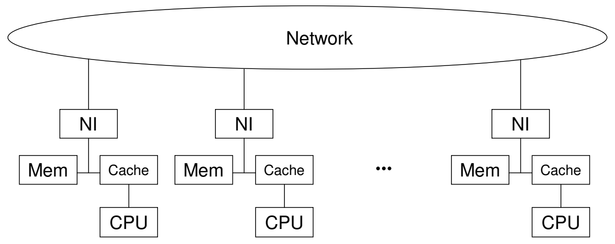

Figure 1: Memory hierarchy of a distributed shared-memory multiprocessor. Memory operations that miss in the cache and that are not stored in local memory are handled by the network interfaces (NI) and generate network traffic, potentially invalidating cache-lines in other nodes.

Next we would like to measure the application speedup for a variety of network parameters, of which the reconfiguration interval will be the most important one. Doing one full simulation, however, can take several hours, so it is not feasible to do this for every combination of parameters and every single benchmark application. Circumventing full simulation by using a different approach, like statistically generating network traffic, is not reliable in this case since no statistical models exist that accurately capture the specific behaviour of network traffic that is exploited by our type of reconfigurable network. Therefore, we have developed a method that can predict application speedup for a range of reconfigurable networks, based on a network traffic trace from just one simulation.

This paper reports on our prediction methodology and analyzes the results. Section 2 describes in more detail the architecture of both the shared-memory machine and the reconfigurable network that were used in this study. In section 3, the methodology that was followed to obtain the communication patterns is described. The prediction method is presented in section 4. Section 5 gives the prediction results and compares them with the actual speedup. 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 to each other. In the extreme end of the loosely coupled family we find examples such as the *Beowulf cluster* [12], 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, mostly 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 them 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 slower 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 ([7], see figure 1). 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).

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 [10, 11]. 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.* A schematic overview is given in figure 2. To simplify routing the extra links are used exclusively by the two linked nodes, multihop transfers use only the base network.

In reality, some limitations will of course apply. Since the number of neighbors for each node is limited (due to maximum optical pin-counts per chip, or the rising complexity of

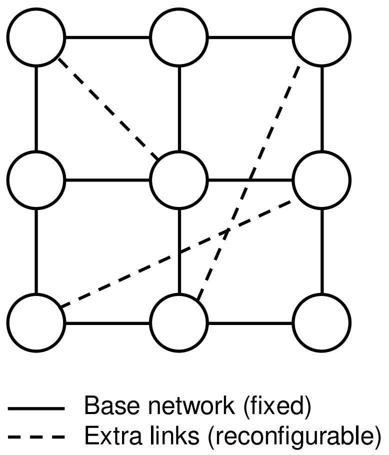

Figure 2: Reconfigurable network topology. The network consists of a base network, augmented with a limited number of direct, reconfigurable links.

a router with a large number of in- and outputs) only a few extra links can at the same time connect to one node, and probably not *every* arbitrary node pair can be connected using only one link. However, for the current study we do not yet take these limitations into account.

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 (figure 3). 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.

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 extra links between the node pairs where communication is the most intense (measured in bytes transferred per fixed-length 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 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 and thus we will need to revise the position of the extra links over time. To this end we would need to know what traffic patterns will be present in the future. This can for instance be done by predicting future patterns based on network traffic from the past, or by annotating the program – based on the result of static analysis of the sharing patterns in the source code, or from measurements of a previous execution of the same program (profiling). For the current discussion however, we assume the presense of a central entity that can

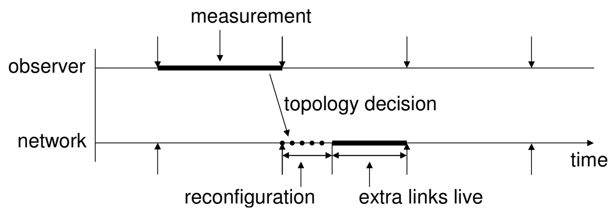

Figure 3: The observer measures network traffic, and at certain intervals makes a decision where to place the extra links. Reconfiguration will then take place, during which time the extra links are unusable.

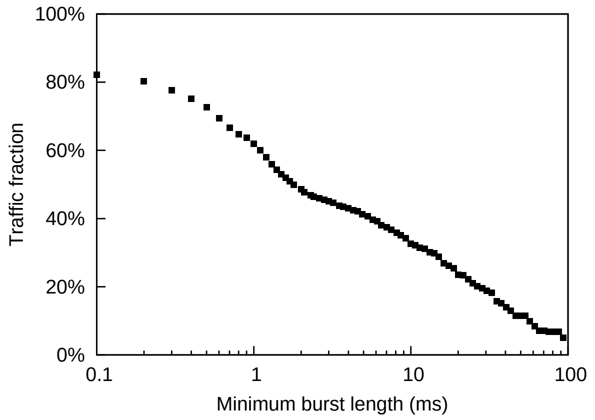

Figure 4: Traffic size fractions per minimum burst length.

make a perfect prediction of the most important node pairs for the next interval.

Because reconfiguration is not immediate (depending on the technology, reconfiguration can take from 100 μs up to several ms), the interval between decision times will be one of the most important parameters. This interval should be long enough to amortize on the cost of reconfiguration, during which the extra links are unusable, but it must be sufficiently short to keep pace with the changing demands made by the application. In [5], we have shown that a significant fraction of the network traffic is part of long bursts of communication (see figure 4 – taken from [5]). This behaviour can be exploited by our type of network. In this paper, we improve the method used to predict application speedup presented in [5] and validate our predictions against actual simulations.

3. METHODOLOGY

We have based our simulation platform on the commercially available Simics simulator [8]. 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 and removed to/from 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 of the base network, yielding an average packet latency that is the same for the 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 representable 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.

Since the simulated caches are not infinitely large, the network traffic will be the result of both coherence misses and cold/capacity/conflict misses. This way, the network traffic is not underestimated as is done in other studies that do not include references to private data in the simulation. 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 [13] 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 to 512 KiB, resulting in a realistic 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 APPLICATION 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 those node pairs that communicate most intensely. Since the switching time is finite, the extra links will stay in place for some larger period of time, called the reconfiguration interval Δt. After every interval of Δt seconds, we predict the communication pattern that will be present in the next interval, and place the n extra links between the node pairs that are predicted to have the most intense communication. We now derive a prediction of application speedup that can be parameterized for n 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 the 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 top n node pairs are found for each interval of Δt seconds.
- The memory accesses that would benefit from the extra links are identified.
- The latency of each memory access is reviewed, for accesses related to the top n this latency is divided by a certain factor.
- 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.

4.2 Full simulation

We start by doing one full-system simulation (per benchmark), as described in section 3. 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 and the destination node.

4.3 Determining the top n node pairs

The packet trace is divided into intervals of Δt seconds. 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[1]. The n node pairs that exchanged the most traffic are stored, and will be further referred to as the *top n node pairs* for this interval. With a perfect traffic predictor, these would be the node pairs between which an extra link is to be made.

When doing the simulation to validate our speedup predictions, in which the extra links are added to the network, it is not possible to have a perfect traffic predictor since this requires non-causal behaviour. Therefore our simulator uses a predictor that uses the top n links from the *previous* interval of Δt seconds. For slowly varying traffic this should give good results, for benchmarks that have highly fluctuating traffic patterns the simulation will of course yield a speedup that is less than that obtained with a better or even perfect traffic predictor.

Instead of blindly using the n links with the most traffic, some optimisations could be made. It is for instance possible that one node is generating heavy traffic to a number of other nodes, this one node will therefore be the endpoint of several extra links. Since the node degree is physically limited (e.g., due to the available off-chip bandwidth or a limitation on the number of wavelengths), this is not a realistic situation. Also it is possible that a node pair in the

[1]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.

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 extra link between them will not improve latency, unless adding this link can significantly decrease congestion. Finally, in our simulation, extra links are only used for traffic that has the same endpoints as the link, not for traffic that might use the extra link as only a part of its path.

Solving these limitations would require topology-generating and routing protocols that are significantly more complex than those in the current case, and may not be compatible with the high-speed low-latency environment inside a shared-memory machine. Therefore we have decided to forego on these issues for now, and delay their study for future work.

4.4 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 top 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 is in the top n for the interval in which the memory access is made, the access is considered to be in the top n. Memory access latencies (a few μs) are significantly shorter than the considered reconfiguration intervals (100 μs 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.

4.5 Calculating new latencies

Traffic that can use an extra link will reach its destination in only one hop, compared to potentially several hops for traffic using the base network. For the 4x4 torus network used in our simulation, the average distance is 2.13 network hops. For traffic between a top n node pair a direct link is used, the extra link, so the number of hops here is always 1. This traffic will therefore, on average, traverse a factor of 2.13 less nodes than traffic using the base network. We now assume that memory accesses between top n nodes will have a latency that is reduced by the same factor[2].

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 make that latency does not always scale 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 in the top n are divided by 2.13, the others are not affected. This relationship remains the same for different values of n and Δt, but different sets of memory accesses will benefit from the extra links.

4.6 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 a fraction of the latter (those memory accesses for which this processor and their home node form a pair that is in the top n) can be sped up.

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, so 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 usefull computations. However, there is usually a significant dependency between instructions, which makes that even a very agressive 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 μs). Therefore out-of-order execution should not influence the results too much.

5. RESULTS AND DISCUSSION

5.1 Results

The predictions and confidence intervals for a number of benchmarks are shown in figure 5. For each benchmark program, 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. All simulations were done using a reconfiguration interval of 1 ms and 16 extra links.

We see that for some benchmarks, our prediction can give a good idea of the attainable speedup. For other benchmarks the average prediction is not good, but the actual

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

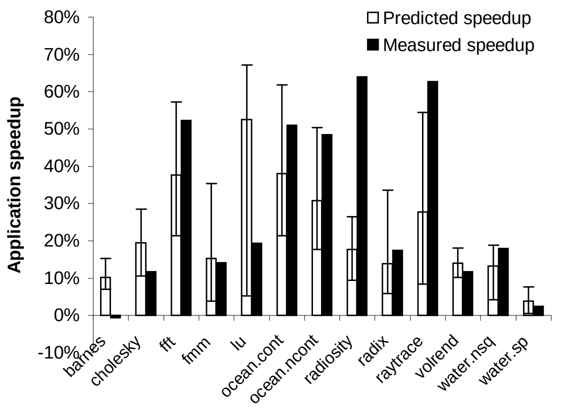

Figure 5: Predicted speedup (with average, upper and lower estimates) and measured speedup for selected benchmark applications.

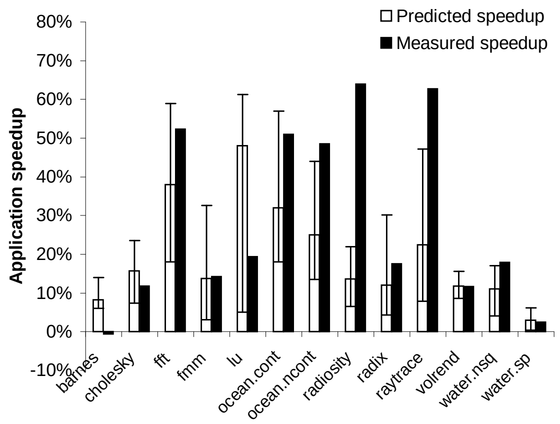

Figure 6: Predicted and measured speedup, here the predictions take the actual traffic predictor into account.

speedup is inside our confidence interval. For three benchmarks (barnes, radiosity and raytrace), the actual speedup is outside the confidence interval. In the following section, we validate our assumptions and try to explain the cause of these errors.

5.2 Validation of assumptions

5.2.1 The traffic predictor

Our speedup predictions were made for a reconfigurable network that could make a perfect prediction of the n most communicating pairs for the next interval. In our simulation however, we had to use a less omniscient traffic predictor, one that uses the top n node pairs from the previous interval as its prediction.

To test the effect of this limited traffic predictor, we have adapted the speedup prediction detailed in section 4 to take the actual traffic predictor into account. This can be done by modifying the third step, in which memory accesses are selected that will benefit from the extra links: we select the accesses that are made between a node pair that is in the *previous* top n, rather than the *current* top n. The rest of the steps are not modified. The predicted speedups are shown in figure 6, and are very similar (within 5%) to the estimated speedups using a perfect traffic predictor. This proves that our limited traffic predictor does a good job and confirms that network load varies rather slowly over time.

5.2.2 Access latency reduction

A second assumption we can evaluate is whether the average link distance is indeed 2.13. It is conceivable that the operating system, or the algorithm used in the benchmark, distribute data such that most data is found on a node closeby. However, this is not the case: we found the average distance spanned by the extra links to be 2.1 ± 0.1 for al benchmarks. Network traffic using one of the extra links will indeed traverse, on average, 2.13 less hops.

The relationship between reducing the number of hops and the reduction in memory access latency is however 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 extra links are added, averaged across all benchmarks). Further analysis of the impact of congestion is therefore necessary, including its distribution across the network and its influence on memory accesses that are or are not on the critical path of the application.

5.2.3 Application variability

Finally, the control path taken by an application can change when the relative timing of the processors is influenced. Some algorithms distribute tasks between processors by using a global task queue. If one processor is delayed because of a memory reference that takes longer to execute, subsequent tasks will likely be assigned to different processors. For some scientific benchmarks, the convergence of computations depends on the order in which tasks are executed. Since changing the relative network latencies can affect this order, it may well affect the rate of convergence of the benchmark, resulting in either a larger or a smaller number of iterations.

To determine whether this effect might be the cause of the largest prediction errors, we have analyzed the benchmark's computation times (the time not spent waiting for memory accesses). The horizontal axis in figure 7 shows the (relative) difference in computation time between the simulation run without extra links (only the base network) and the simulation with extra links. The vertical axis shows the prediction error (measured runtime versus predicted average processor runtime). We clearly see that there is a large correlation, meaning that most of the misprediction is due to the application following a different control path before and after adding the extra links to the network.

For applications that exhibit such behaviour, it is not possible to accurately estimate application speedup in this way. Several simulations would need to be done to determine the

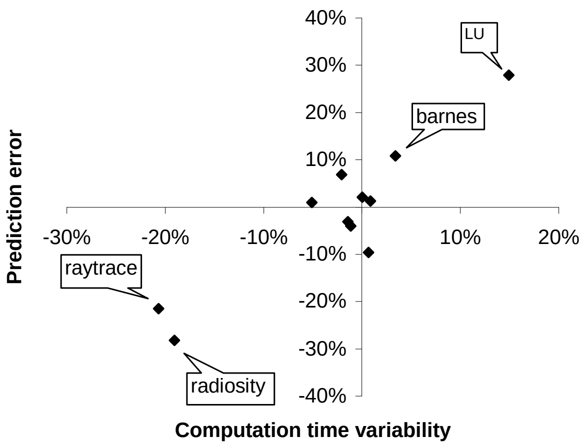

Figure 7: Correlation between speedup prediction error and variability in computation time.

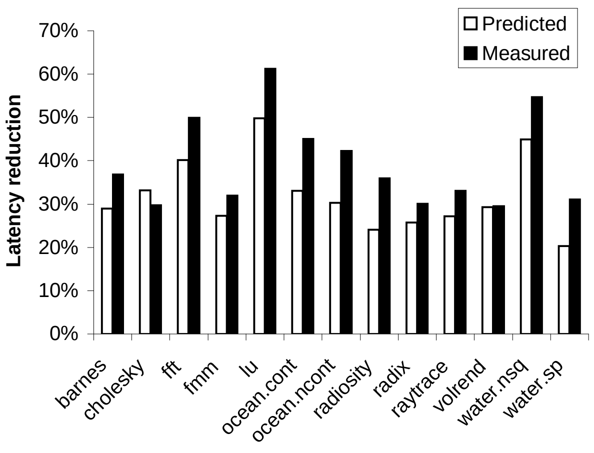

Figure 8: Reduction in average remote memory access latency, predicted and measured values (average reduction: 40.3%, average predicted reduction: 32.4%).

variability, and an average speedup can be computed with a certain confidence interval. Alternatively, one could only predict the average memory access latency, without translating this to application speedup. This can be done with our technique, and is much less influenced by application variability. Whereas execution speedup remains the most important metric, the reduction in access latency can certainly be used as a performance metric for comparing different network parameters.

Figure 8 shows the average reduction in remote memory access latency, both predicted and measured. Averaged across all benchmarks, the average latency is reduced from 1215 ns to 821 ns (predicted) or 725 ns (actual). The systematic underprediction is mainly due to the fact that our prediction does not take the reduction of congestion into account.

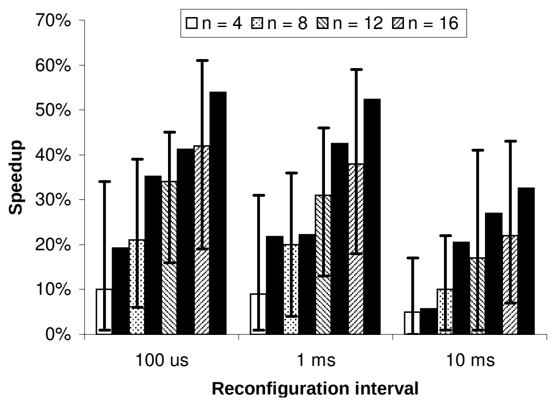

Figure 9: Predicted and measured speedup for the FFT benchmark. Results are shown for reconfiguration intervals (Δt) of 100 μs, 1 ms and 10 ms, and 4, 8, 12 and 16 extra links (n).

6. FUTURE WORK

Several assumptions were made in the prediction model, they have been stated throughout section 4. Some of them are fairly trivial, others need more work to either validate or invalidate them. We will also attempt to include congestion effects in our prediction model.

As discussed in section 4.3, there are a number of shortcomings to the approach of using the top n node pairs without further discrimination. A few improvements were proposed, such as limiting the degree of each node and considering the gain of a path using the extra link compared to the path over the base network. Both of these relate to the underlying architecture of the reconfigurable network, and require extensions to our simulator as well as our predictor.

Finaly, we will validate our prediction model for a wider range of n and Δt and other parameters, and use it to explore the design space for reconfigurable interconnect networks. Some preliminary measurements for the FFT benchmark are given in figure 9, showing that our prediction model can give good results over the considered range of parameters.

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 network latency and total runtime for variable network parameters (number of extra links n, reconfiguration time Δt, link latency, ...) based on a single simulation run per benchmark and per base network configuration. We have also defined confidence intervals for our prediction and found that, for most benchmarks, predicted runtime speedups fall within those intervals. We have analyzed the impact of several assumptions made in our model, and found the benchmark control flow itself to be responsible for the largest errors. Most of the remaining errors (an almost systematic overestimation of memory access latency) are probably due to congestion reduction, which is not yet incorporated in our model. Future work will be aimed at improving the link allocation method and adapting our prediction model accordingly. We will also attempt to include congestion effects in our prediction model.

8. 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.

9. 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*, volume 4455, pages 160–171, Bellingham, July 2001.

[2] K. Bui Viet, L. Desmet, J. Dambre, K. Beyls, J. Van Campenhout, and H. Thienpont. Reconfigurable optical interconnects for parallel computer systems: design space issues. In *VCSELs and Optical Interconnects*, volume 4942, pages 236–246, Brugge, October 2002. SPIE.

[3] L. Chao. Optical technologies and applications. *Intel Technology Journal*, 8(2), May 2004.

[4] 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*, 39:671–682, 2000.

[5] W. Heirman, J. Dambre, J. Van Campenhout, C. Debaes, and H. Thienpont. Traffic temporal analysis for reconfigurable interconnects in shared-memory systems. In *Reconfigurable Architectures Workshop*, April 2005.

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

[7] 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*, 25(3):63–79, March 1992.

[8] 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*, 35(2):50–58, February 2002.

[9] 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*, 41(1):42–52, 1997.

[10] T. M. Pinkston and J. W. Goodman. Design of an optical reconfigurable shared-bus-hypercube interconnect. *Applied Optics*, 33(8):1434–1443, 1994.

[11] 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.

[12] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf : A parallel workstation for scientic computation. In *Proceedings of the International Conference on Parallel Processing, Boca Raton, USA*, pages 11–14. CRC Press, August 1995.

[13] 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*, pages 24–36, Santa Margherita Ligure, Italy, 1995.