Photon Netw Commun (2008) 15:25–40
DOI 10.1007/s11107-007-0084-z

Predicting the performance of reconfigurable optical interconnects in distributed shared-memory systems

Wim Heirman · Joni Dambre · Iñigo Artundo · Christof Debaes · Hugo Thienpont · Dirk Stroobandt · Jan Van Campenhout

Received: 5 July 2007 / Accepted: 25 July 2007 / Published online: 12 September 2007

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, a tool commonly used when doing a fast design space exploration. In this paper, we present a technique that can 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. This again allows for a quick comparison of different network implementations with good relative accuracy, narrowing down the design space for more detailed examination.

Keywords Prediction · Reconfiguration · Interconnect · Shared-memory

1 Introduction

1.1 Background

The 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 bus load, 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 [5,6] 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 run-time reconfigurable interconnection network [7,8] 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 intensively have a direct connection. Since there is no longer interference from other traffic streams, this results in higher available bandwidth and lower packet latency (which is otherwise caused by congestion and switching delays).

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

W. Heirman (✉) · J. Dambre · D. Stroobandt · J. Van Campenhout
ELIS Department, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium
e-mail: wim.heirman@elis.ugent.be

I. Artundo · C. Debaes · H. Thienpont
TONA Department, Free University of Brussels, Brussels, Belgium

temporal behavior of the interprocess data transfer patterns. In our previous work, we have characterized the locality in both time and space of the traffic flowing over the network [9], using full-system simulations 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 node pairs, and that the particular node pairs between which communication is done, varies over time. This suggests that reconfigurable networks can result in a significant performance improvement. Next, we have included the model of a specific reconfigurable network in our simulator, and found that the average remote memory access latency could be improved by up to 20% [7].

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 thousands of simulations. In a full-system simulation the entire machine is modeled, including processors, caches, memories, and the interconnection network. This virtual machine runs both the operating system and the benchmark, allowing the traffic on the network to be driven by an actual benchmark application (hence the name *execution-driven*). Due to its complexity, for realistic benchmarks one such simulation can take several days to complete. It is therefore 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 [11]. 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. This precludes their use for our purposes, leaving us with the much slower full-system simulations.

Another well-known technique, called *trace-driven simulation*, is situated between full-system simulation and statistical traffic generation. Trace-driven simulation starts with one full-system simulation, where the resulting traffic flow is recorded. This 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.

1.2 Paper contributions

In this paper we present a fast prediction method for the performance of reconfigurable networks. It 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 manifests 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 new average remote memory access latency. This will be our performance metric for the network. With this quick estimation method a network designer can explore the design space of the interconnection network and make a comparison between different proposed network architectures. Note that this only requires a good *relative* accuracy of our model, an *absolute* prediction of network performance is usually not necessary at this stage.

In Sect. 2 we situate this work in relation to other research on reconfigurable and optical interconnections. Sect. 3 describes in more detail the architecture of both the shared-memory machine and the reconfigurable network that were used in this study. In Sect. 4, relevant details of our simulation setup are provided. The prediction method is presented in Sect. 5. Section 6 gives the prediction results and compares them with the actual speedup. In Sect. 7 some future work is discussed, the conclusions are summarized in Sect. 8.

2 Related work

Several papers present optical interconnection demonstrators for point-to-point inter-chip interconnects. Ref. [12] describes the result of the OIIC project performed in cooperation with our own research group, in which two FPGA chips were connected directly using 2D arrays of optical components, flip chip mounted onto the FPGA chips. Ref. [6] describes a similar project performed at IBM Research, while Ref. [5] contains several articles about the current state of Intel's research in optical interconnections.

At the systems level, Collet [3], Huang [13] and Benner [4] each explore the potential of optical interconnections in several types of computing systems. All agree that optics can be used effectively in the class of tightly coupled, medium to large scale SMP systems.

On the reconfigurability side, early work done by Snyder introduces the reconfigurable computer [14]. He notes that an adaptive network can make a parallel computer achieve good performance for a range of applications, each possessing different communication patterns. Pinkston describes the GLORI system [15], a possible implementation of an architecture similar to Snyder's, using optical reconfiguration technology. Other architectures, some of which have been developed into demonstrators, include the simultaneous optical multi-processor exchange bus (SOME-bus) from Drexel University [8], the optical centralized shared bus from the University of Texas at Austin [16], and Columbia University's vortex optical packet switching interconnection network [17].

The work done by our research group tries to combine several aspects of these works: the view, at a systems level, of reconfigurable optical interconnections applied to large parallel computer systems. In Ref. [18] we gave an overview of the technologies and ideas that are applicable, and presented a number of simulations showing the results that can be expected with this technology. We also reported on a prediction method that allows fast design-space exploration [19]. However, due to a desire to keep complexity of the network implementation down, this work imposed a restriction on how the reconfigurable network was used: a network packet either had to go through an extra link, or through the base network, but not through a combination thereof.[1] In some implementations, one being our own proposal described in Ref. [18], it is impossible for some node pairs to be connected by an extra link. Traffic exchanged between this node pair will therefore never benefit from the reconfigurability of the network, limiting the performance that can be attained.

This paper extends on the preliminary prediction method from Ref. [19]. By adding more intelligence to the algorithms controlling reconfiguration, the routing restriction has been removed so that packets can use a combination of base network and extra links. This allows for a much more efficient use of the reconfigurable network, increasing performance of the parallel computer. For our prediction method, this change required us to review latency on each memory access separately, instead of being able to use a global improvement factor as in [19]. This somewhat increased both the implementation complexity and the runtime of our prediction method, but also improved the prediction accuracy.

3 System architecture

3.1 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 (Fig. 1). In the extreme end of the loosely coupled family we find examples such as the *Beowulf cluster* [20], in which the network consists of a commodity technology such as Ethernet. This simple interconnection network results in relatively low throughput (1 Gbps per processor) and high latency (up to several milliseconds, 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.

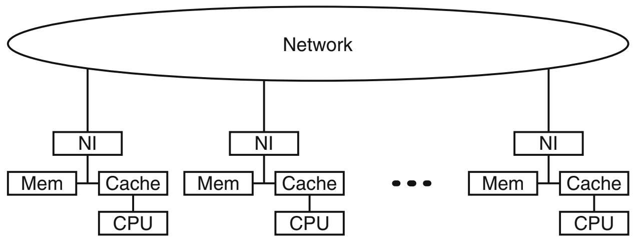

Fig. 1 Schematic overview of a multiprocessor machine. For message-passing machines, the network traffic is under control of the application. In shared-memory machines, network traffic is generated by the network interfaces (NI) in response to non-local memory accesses by a processor

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 sub-problems (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. However, 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 the latter class of machines range from small, 2- or 4-way SMP server machines (including multi-core processors where several 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. Since the capabilities of electrical networks are evolving much more slowly than processor speeds, reconfigurable optical interconnection networks are a likely alternative for large shared-memory systems.

For this study we consider a machine in which coherence is maintained through a directory-based coherence protocol.

[1] The notion of *extra link* is introduced in Sect. 3.2, where we describe our reconfigurable network architecture.

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 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 consists 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).

Remote memory access, where use of the interconnection network is required, 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 to hide some of the communication latency, but to use this technique to speed up one parallel program would require the program to be split up into more threads. While communication between threads running on the same processor core is essentially free, synchronization is not so multithreading again increases overhead. Ultimately, it is very difficult to effectively hide the communication latency, which makes system performance very much dependent on network latency.

3.2 A simple reconfigurable network architecture

Previous studies concerning reconfigurable networks, such as [15] and [21], 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. 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. Moreover, implementing this kind of network usually requires large free-space optical structures which are not compatible with the integrated, highly reliable nature of large parallel computers. Finally, the communication pattern exhibited by the program ran on the parallel computer may have a structure that cannot be mapped efficiently (i.e., using only single-hop connections) on the chosen base network topology, suggesting that reconfiguration should be used to offer a richer topology than that of the base network.

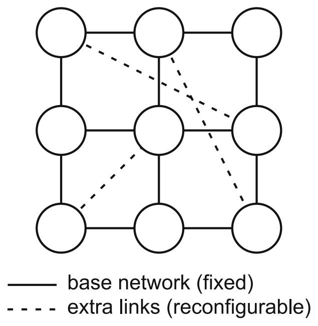

Fig. 2 Reconfigurable network topology. The network consists of a base network, augmented with a limited number of direct, reconfigurable links

Therefore, we assume a different network architecture in this study. We start from a base network with a 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 Fig. 2. An advantage of this setup, 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. 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.

The reconfiguration of our network aims to exploit the temporal locality of the communication patterns. 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 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 (Fig. 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 instantly. This is not the case in reality: depending on the technology, reconfiguration (the *switching time*) can take from tens of microseconds up to several milliseconds. Therefore, the

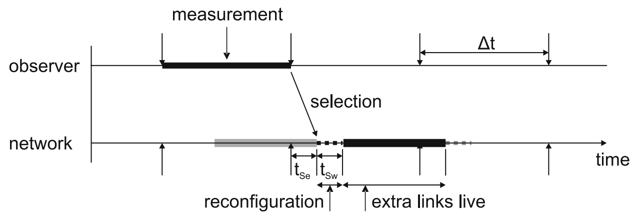

Fig. 3 The observer measures network traffic, and after each interval of length Δt makes a decision where to place the extra links. This calculation takes an amount of time called the *selection time* (t_{Se}). During the *switching time* (t_{Sw}), reconfiguration will take place making the extra links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

reconfiguration interval must be chosen such that it is large compared to the selection and switching times. 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.

3.3 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 Cavitiy 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. Note that the routing to and from the broadcast element will be such that nodes have different neighbors on the broadcast element than those on the base network. This way the extra links will span a distance on the base network that is larger than 1.

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 (non-reconfigurable) optical interconnects are currently arriving to the midrange servers. More information about this envisioned implementation can be found in Ref. [18].

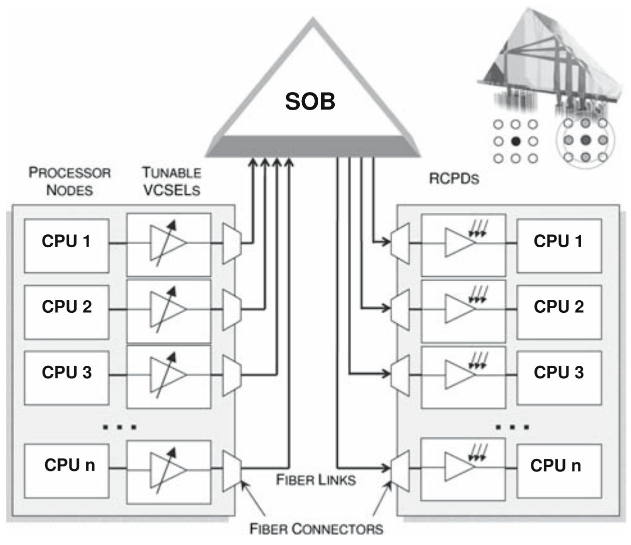

Fig. 4 Schematic representation of an example reconfigurable optical interconnect implementation. A processor node transmits data on one of nine wavelengths $\lambda_1 \ldots \lambda_9$. The selective optical broadcast element (SOB) distributes the signal toward nine fellow processor nodes. Since every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

Note that while optical interconnections (light source $\rightarrow$ waveguide $\rightarrow$ detector) are unidirectional, elinks as defined in this work are bidirectional. Therefore, an elink consists of two such assemblies. 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.

4 Methodology

4.1 Simulation platform

We have based our simulation platform on the commercially available Simics simulator [10]. 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 (2 cycles access time for L1 caches, 19 cycles for L2 and 100 cycles for SDRAM). The directory-based coherence 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 coherence 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).

To avoid deadlocks, dimension routing is used on the base network. Each packet can go through one elink on its path, after that it switches to another virtual channel (VC)[2] to avoid deadlocks of packets across elinks. 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 KB 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 private data for the duration of the program.

The SPLASH-2 benchmark suite [23] was chosen as the workload. It consists of a number of scientific and technical applications using a multi-threaded, shared-memory programing model. Since the default benchmark sizes are too big to simulate their execution in a reasonable time, smaller problem sizes were used (Table 1). Since this influences the working set, and thus the cache hit rate, the level 2 cache was resized from an actual 8 MB on a real UltraSPARC III to 512 KB. Also the associativity was increased to 4-way (compared to 2-way for the US-III) after we experienced excessive conflict misses in Solaris' internal structures with the 2-way caches. Overall, this resulted in 93–97% hit rates for the L2 caches. 50–60% of L2 misses were cataloged as coherence misses (resulting in communication between different processors), the remaining 40–50% were cold/conflict/capacity misses.

The simulation slowdown (simulated time versus simulation time) was a factor of 50,000 resulting in execution times of 1–10 h per benchmark on a Pentium 4 running at 2.6 GHz with 2 GB RAM. In contrast, the prediction method described later in this paper, which seeks to replace a large percentage of these slow simulations during a design-space exploration, only requires about 1–10 min of computation time.

4.2 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 time between reconfigurations, called the reconfiguration interval Δt, is the third parameter. The results in this paper will be based on different sets of values for these three parameters. Additionally, results for a network using the selective optical broadcast element described in Sect. 3.3 will be shown as illustration of the performance of an actual implementation. This network can be modeled using $n = 16$, $f = 1$, and additional constraints on which destinations (only 9 out of 16) can be reached from each source node.

4.3 Extra link selection

For every reconfiguration interval, a decision has to be made on which elinks to activate, within the constraints imposed by the architecture, and based on the expected traffic during that interval (with, in our current implementation, the expected traffic being the traffic as measured during the previous interval). As explained in Sect. 3.2, we want to minimize the number of hops for most of the traffic. We do this by minimizing a cost function that expresses the total number of network hops traversed by all bytes being transferred. This cost function can be written 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 elinks are bidirectional elinks, $T(i, j)$ is the sum of traffic in both directions.

The time available to perform the extra link selection is from the same order of magnitude as the switching time, because both need to be significantly shorter than the reconfiguration interval. Since the switching time will typically be in the order of milliseconds, we need a fast heuristic that can quickly find a set of active elinks that satisfy the constraints imposed by the architecture and has an associated cost close

[2] Actually another *set* of VCs is used since we already employ separate request and reply VCs to avoid *fetch deadlock* [22] at the protocol level.

Table 1 SPLASH-2 benchmark applications and their problem sizes that were used throughout this paper

Code	Problem size
Barnes	8K particles
Cholesky	tk15.O
FFT	256K points
Ocean-Cont	258 × 258 ocean
Radix	1M integers, 1024 radix

```
nodepairs = [ all (src, dst) pairs with src < dst]
nodepairs.sort(
  sort_by = distance_using_basenetwork(src, dst)
            * traffic(src, dst),
)

active = []
possible = [ all elinks supported by the architecture ]

while not nodepairs.empty() and not possible.empty():
  (src, dst) = nodepairs.pop()
  elink = most_interesting_elink(src, dst)

  if distance_using_elinks(src, dst, active + elink)
     >= distance_using_elinks(src, dst, active):
    # no additional gain by turning on elink
    continue

  # turn on elink!
  possible.remove(elink)
  active.add(elink)

  # make sure we obey all implementation constraints,
  #   such as maximum fanout
  for elink in possible:
    if conflicts(active, elink):
      possible.remove(elink)

activate_elinks(active)
```

Fig. 5 Pseudocode for the elink selection algorithm

to the global optimum. We have constructed a greedy algorithm that works as follows (see Fig. 5 for an implementation of the algorithm in pseudocode):

1. A list is constructed of all node pairs (i, j), sorted by $d(i, j) \cdot T(i, j)$ in descending order, with $d(i, j)$ the distance between nodes i and j when using only the base network connections.
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), for the implementation with the selective broadcast element from Fig. 4 E_p will contain a subset hereof.
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 these two 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, since it reduces the distance to 1. 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} × {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 following iterations of the algorithm. 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 will be enabled during the next time interval.

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 that results in the global minimum of C. This takes however several minutes to execute for each given traffic pattern, which underlines the need for a fast heuristic: actual reconfiguration times (including elink selection) will be in the order of milliseconds. In a simulated environment this does not matter of course, so we ran some simulations to see the difference in latency

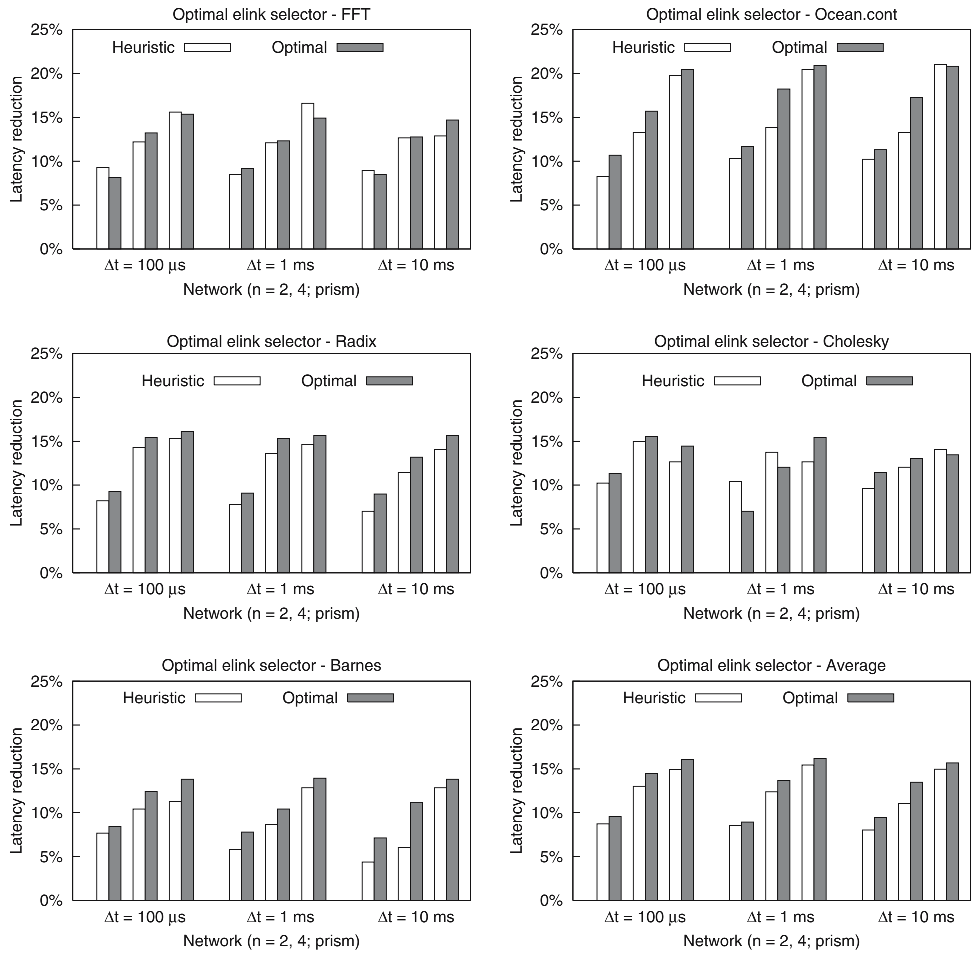

Fig. 6 Comparison of the heuristical elink selection algorithm and the globally optimal elink placement

improvement between this optimal selector and our heuristic (Fig. 6). Results are shown for three reconfiguration intervals and three different network implementations: two in which two or four elinks can be placed freely (with an imposed fanout limit of two) and the *prism* scenario which uses the implementation with the prism from Sect. 3.3. In a few cases, the optimal elink placement results in a *slower* network than the pseudo-optimal placement from our heuristic. This is because the elink placement is only optimal for the traffic pattern from the *previous* interval, the traffic in the *current* interval may have shifted such that the heuristical selector now gives a better result. In most cases however, the result is as expected, with the network using the optimal selector being a few percent faster. This means our heuristic does a good job and only slightly affects network performance.

5 Predicting network performance

5.1 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 constraints imposed by the network (Δt, n and f, or the properties of the selective broadcast element from Sect. 3.3), and can therefore predict the performance of a range of candidate networks, while still relying on only a small number of long running simulations. For each benchmark, this prediction is derived using the following steps:

- A single full simulation is done of each benchmark, using a non-reconfigurable network (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 Δt.
- The placement of the extra links, given the traffic pattern just computed, is determined for each interval using the algorithm described in Sect. 4.3.
- 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 latency reduction is derived.

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

5.2 Full simulation

We start by doing one full-system simulation (per benchmark), using the simulation platform described in Sect. 4.1. Only the base network is active, so this simulation also serves as the baseline against which we calculate the speedup to determine the performance 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 and destination nodes and the packet size. Note that there are only two sizes of messages generated by the coherence protocol: 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.

5.3 Determining the extra link placements

The packet trace is divided into intervals of duration 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 extra links are bidirectional, so traffic in both directions is added together. When we have the traffic profile for the interval, we use the greedy algorithm described in Sect. 4.3 to determine extra link placements for the next interval.

5.4 Correlating memory accesses

The metric that makes network performance visible to the processors, is the remote memory access latency. Therefore, we have chosen the relative memory access latency reduction (compared to the baseline memory latency) as the metric with which to compare different networks. We will now estimate the new memory access latency as a function of the selected elinks.

We enumerate the memory accesses of the execution and represent each access by its sequence number $i \in \{1, 2, \ldots, m\} = I$. 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 i with these two distances: its *baseline distance* $d_\mathrm{b}(i)$ and its *elink distance* $d_\mathrm{e}(i)$.

Memory access latencies (taking at most a few μs) are significantly shorter than the considered reconfiguration intervals (100 μs and upwards), so there should be no problem 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.

5.5 Calculating new latencies

First we calculate the average memory access latency, over the course of the baseline simulation, for all memory accesses with the same distance (with $L_\mathrm{b}(i)$ the baseline latency of memory access i, and $\overline{x(y)}|_{\mathrm{range}(y)}$ denoting the average of $x(y)$ over the specified range of y):

$$L(d) = \overline{L_\mathrm{b}(i)}|_{\{i\,:\,d_\mathrm{b}(i)=d\}}$$

The average baseline access latency can be computed as a weighted average of these per distance latencies, with the number of accesses per baseline distance $N_\mathrm{b}(d)$ as the weights (N is the total number of remote memory accesses in the simulation):

$$L_b = \overline{L_\mathrm{b}(i)}|_I = \frac{1}{N}\sum_d N_\mathrm{b}(d) \cdot L(d)$$

$L(d)$ gives us an estimated memory access latency as a function of the distance between its primary nodes. As indicated by the notation, we assume latency is *only* a function of this distance, and does not change when adding reconfigurable extra links to the network. Therefore, the predicted access latency after adding elinks, $\hat{L}_\mathrm{e}$, would be the $L(d)$ associated with the elink distance d_e:

$$\hat{L}_\mathrm{e}(i) = L\left(d_\mathrm{e}(i)\right)$$

Note that we will always use $\hat{L}$ to denote the value of L as estimated by our model, a measurement of L using simulation will be written as L. Our estimate for the average memory access latency after adding the elinks, $\hat{L}_\mathrm{e}$, can now be computed. We can again count the number of accesses per elink distance $N_\mathrm{e}(d)$ to compute the new average as a weighted average of $L(d)$ using $N_\mathrm{e}(d)$ as weights:

$$\hat{L}_\mathrm{e} = \overline{\hat{L}_\mathrm{e}(i)} = \frac{1}{N}\sum_i L\left(d_\mathrm{e}(i)\right) = \frac{1}{N}\sum_d N_\mathrm{e}(d) \cdot L(d)$$

When written this way, it can be seen that our prediction will be accurate if (1) the new distance distribution $N_\mathrm{e}(d)$ can be

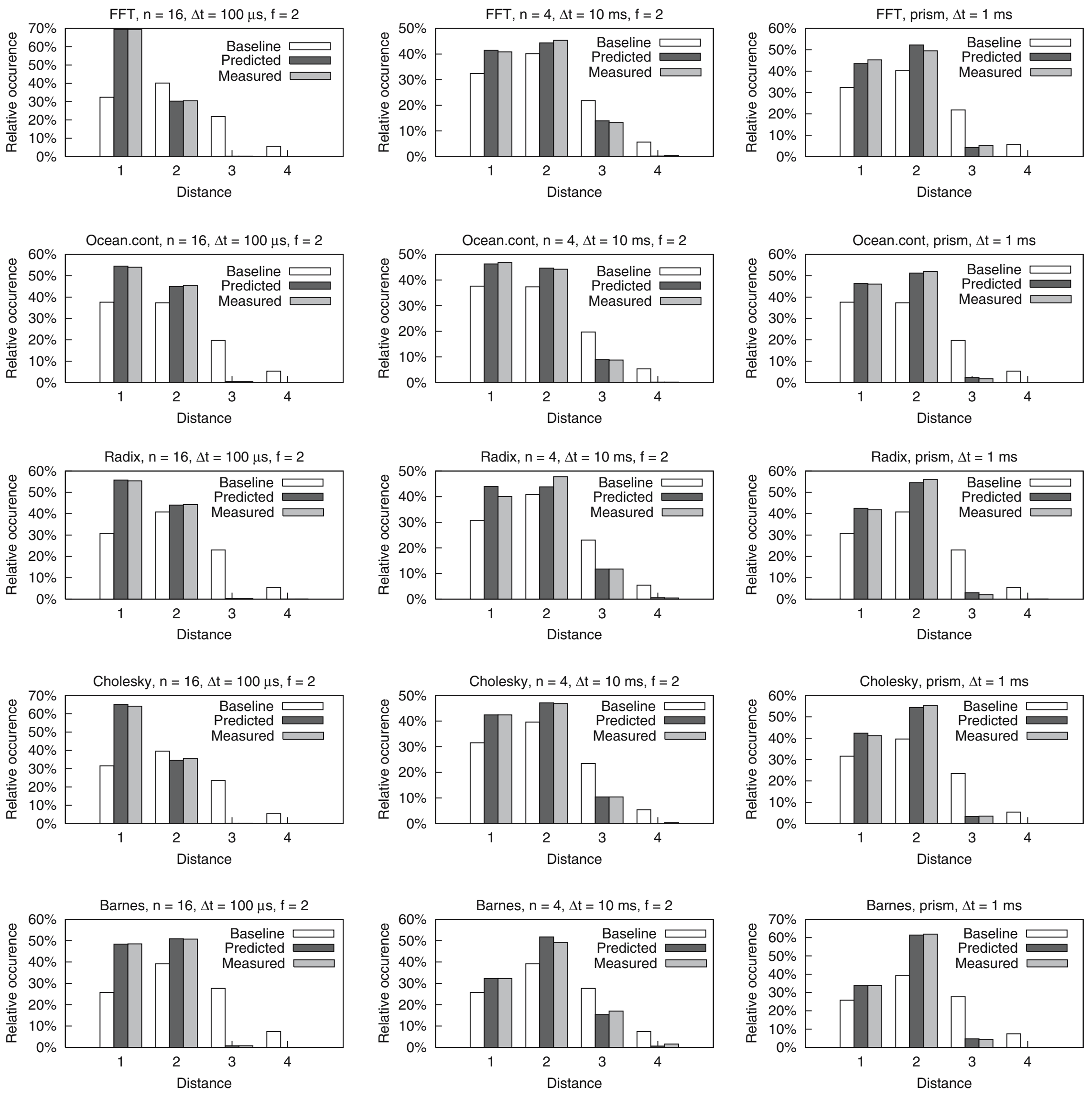

Fig. 7 Baseline, estimated and actual distance distribution of memory operations for FFT, Ocean.cont, Radix, Cholesky, and Barnes benchmarks and a selection of networks

accurately predicted and (2) the per distance latency $L(d)$ does not change after adding elinks.

Figure 7 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 N_b, the predicted number of accesses N_e 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 distribution 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 8 shows this memory latency $L(d)$ for the baseline (bars) and a few different networks (crosshairs), as measured in simulations with the reconfigurable network in place. For $d = 4$ the difference is obvious. However, $d = 4$ accesses are almost eliminated after adding elinks (as can be seen in Fig. 7, where the gray and black bars for $d = 4$ are barely visible), making this measured average (and the apparent rise in latency for most of the networks) unreliable. Moreover, since the number of $n = 4$ accesses, and thus the weight of $L(4)$ in the computation of $\hat{L_e}$, is so small, the value of $L(4)$ does not influence the result much. Lower distances show far less variation, the dif-

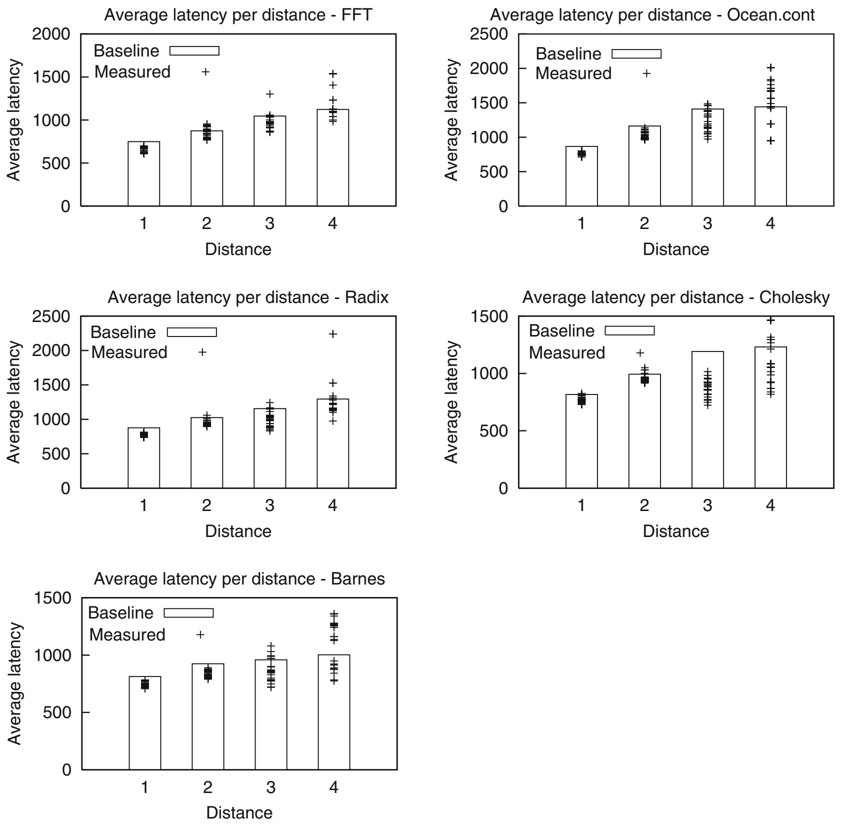

Fig. 8 Variation of average memory latency per distance for different network parameters

ference being due to the reduction of congestion after adding more links to the network. This congestion has not been modeled further, and will be treated as an error term in our prediction.

6 Results and discussion

6.1 Results

Figure 9 shows the result of our prediction model, the memory access latency improvement over the baseline after adding the extra links, compared to the values measured in an execution-driven simulation, for a number of different reconfigurable networks. A linear regression of the form $\hat{P} = \alpha + \beta \times M$ is calculated (with P and M the predicted and measured latency reduction, respectively). The correlation coefficient r is high, so a strong, linear correlation exists between measurement and prediction. Our method can therefore be used to very quickly compare different proposals for network parameters. This makes it a very useful tool for design-space explorations, where the optimum solution needs to be found from a large collection of candidate networks.

In Fig. 10 the predicted (white) and measured (light gray) latency improvements are shown again. This is done for a number of extra links ($n = 2, 4, 8$ and for the implementation using the prism from Sect. 3.3) and different reconfiguration intervals ($\Delta t = 100\,\mu$s, 1 ms, 10 ms). For comparison, the 'fixed' case is added: here, the same number of elinks is added in random positions (favoring longer links), but they are not reconfigured at runtime (the average result from five different placements is reported).

From this graph it is obvious there is a systematic underestimation present. This is most likely due to the fact that we have not included congestion in our method. However, the *relative* prediction accuracy between different network parameters, which is actually the most important value when comparing different suggested network implementations, is much better. By using one more execution-driven simulation, per benchmark, to 'calibrate' our model, better absolute accuracies can be obtained. The third, dark gray bars in Fig. 10 show this calibrated data. They are obtained by scaling the predictions with a constant factor, dependent only on the benchmark application, such that the prediction for a network with parameters $n = 16$, $\Delta t = 100\,\mu$s, $f = 2$ would match the measured value. We chose a network with a large latency improvement for this, so as to minimize the influence of the offset of our predictions (the α parameter in Fig. 9).

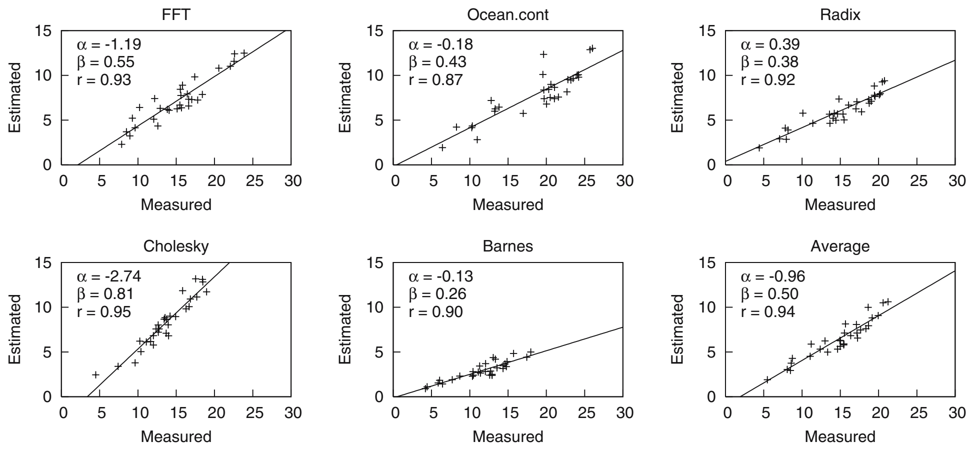

Fig. 9 Estimated versus measured latency reduction for a variety of network implementations. α, β, and r represent a linear regression of the form $\hat{P} = \alpha + \beta \times M$ and its correlation coefficient

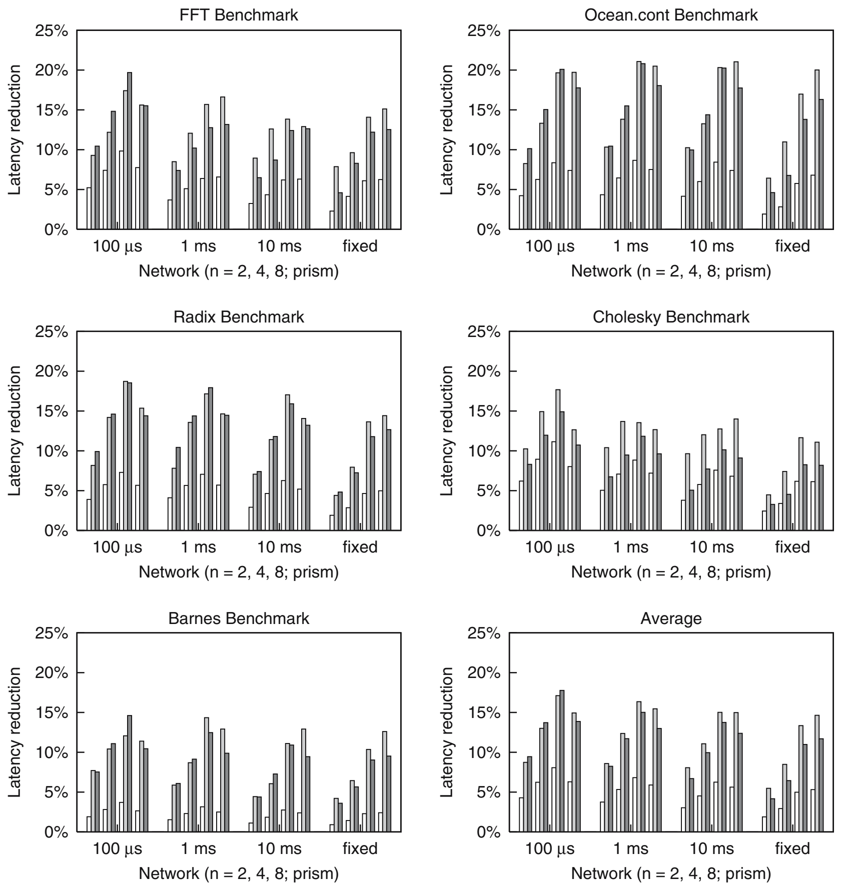

Fig. 10 Latency improvement after adding elinks: estimated using our predictor (white), measured in simulation (light gray), and again the prediction with a corrected factor applied (dark gray)

6.2 Improving accuracy

A number of assumptions were made in the algorithm described in Sect. 5. First of all, the traffic pattern from the baseline simulation is used, when adding elinks traffic streams could potentially appear or disappear. Since the communication pattern is the result of an algorithm that is implemented by the benchmark code, which is in most cases unaffected by the platform on which it is running, this pattern should not change too much. Selecting the elinks based on the traffic pattern is done using the same method as that used at runtime, so here no additional error can be introduced. Figure 7 showed that we can very accurately predict the distance distribution of memory operations after adding a reconfigurable network. Memory latency is however, as Fig. 8 clearly shows, not just a function of hop distance, as was assumed in Sect. 5.5.

The component which causes the largest error in our prediction is 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

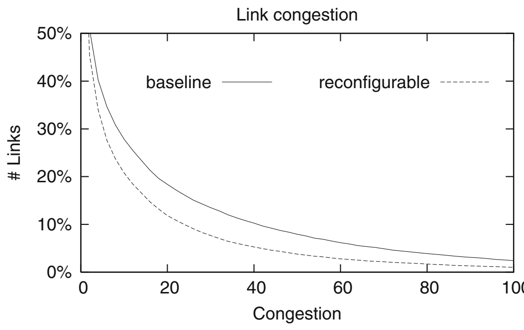

Fig. 11 Complementary cumulative distribution of congestion over the links (both base network links and elinks), for the baseline simulation and a reconfigurable network simulation (with $n = 4$, $\Delta t = 100\,\mu$s and $f = 2$). Congestion of a link is computed as the aggregate time packets need to wait in a buffer before entering the link in question, and is plotted in an arbitrary unit

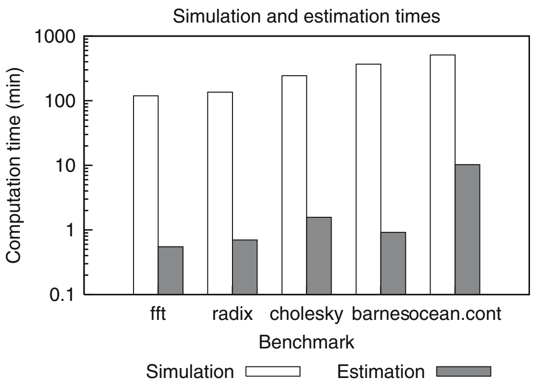

Fig. 12 Computation time (in minutes) required for a full simulation (white) and our prediction (black) for four of the benchmarks considered. Note that the Y-axis is in a logarithmic scale

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. This can be clearly seen in Fig. 11 which shows the complementary cumulative distribution of congestion over the links: for the baseline the line drops off slower meaning there are more links with higher congestion.[3] Further analysis of the impact of congestion is therefore necessary; we leave this for future work.

6.3 Reduction in simulation time

Figure 12 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 simulations in the computation time for our method, since this should only be done once and can subsequently be reused for any number 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 an in-order processor model. 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.

6.4 Effects of traffic prediction

In Sect. 3.2, it was noted that to place the elinks an estimate is needed of what traffic is to be expected during the current reconfiguration interval. Our implementation assumes this traffic will be equal to that measured during the previous interval. We can now ask the question if this assumption is valid, and, if this turns out not to be the case, what increased performance can be expected of a system where traffic can be predicted more accurately. To answer this question through normal simulation we would have to develop such a traffic predictor first, which is not trivial. On the other hand, with our performance prediction method an upper bound can be determined for the performance of a system with an ideal traffic predictor: since, while running the performance prediction, the execution-driven simulation yielding all network traffic has completed, all traffic is known, including that of the ‘current’ interval. So in 5.3, where we determine the elink placement for a certain interval, we can use the traffic for the current interval instead of that for the previous interval. This mimics the behavior of a system with perfect traffic prediction. The results of this change are shown in Fig. 13. As expected, the ‘current’ case, where elinks are placed at locations ideal for the current traffic, performs consistently better than the ‘previous’ case which uses the realistic placement based on past traffic. The difference is however not dramatic, which shows that our assumption that traffic does not change significantly between reconfiguration intervals holds. Also, since the ‘current’ case represents the upper limit for all traffic prediction methods this means that even a very sophisticated and expensive traffic prediction algorithm can only result in a very limited performance increase.

7 Future work

Several assumptions were made in the prediction model, they have been stated throughout Sect. 5. Some of them are fairly trivial, others need more work to either validate or invalidate.

[3] We define the congestion of a link as the aggregate time packets need to wait in a buffer immediately preceding this link before being transmitted over the link. It would be in a $time \times numberofpackets$ unit, but for the graph it was rescaled to an arbitrary range.

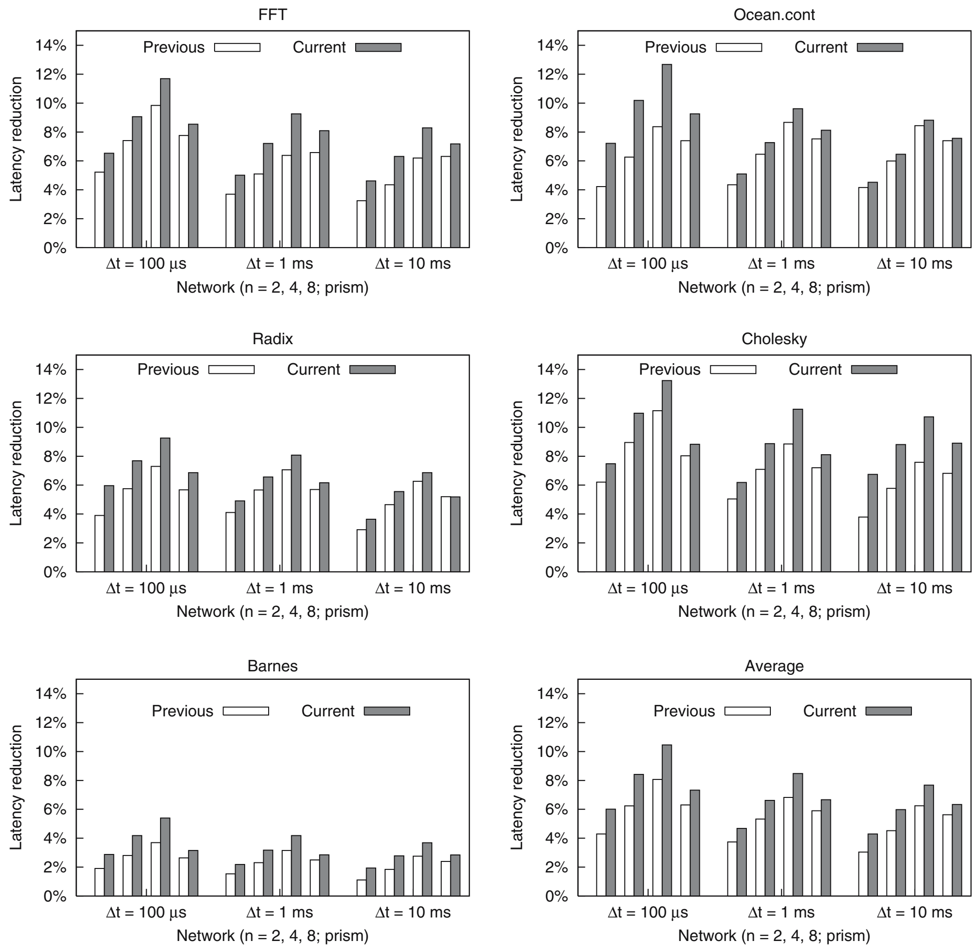

Fig. 13 Performance estimate for a system with a perfect prediction of the traffic in the current interval: in 'previous', elinks are placed at positions ideal for traffic measured in the previous reconfiguration interval, in 'current' elinks are placed ideally suited for the traffic in the current interval

Most importantly, the inclusion of congestion effects into our model should be our next goal.

Our prediction method can also be used to tune other parameters than the n, f, and Δt we have explored in this work. Indeed, we were able to compare the implementation using the selective optical broadcast element with the generalized network described by n and f, other implementations imposing different constraints on the elink placement can be examined also. Finally, the elink selection algorithm, which is part of our prediction model, can be modified. This way we can quickly measure the effect of changes in the selection heuristic.

8 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 remote memory access latency for variable network parameters (number of extra links n, node fan-out f, reconfiguration time Δt) based on a single detailed simulation per benchmark. We found that relative performance over different network parameters can be accurately predicted. We 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.

Acknowledgments This paper presents research results of the Inter-university Attraction Poles Programs PHOTON (IAP-Phase V) and photonics@be (IAP-Phase VI), 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] Miller, D.A.B., Ozaktas, H.M.: Limit to the bit-rate capacity of electrical interconnects from the aspect ratio of the system architecture. J. Parallel Distributed Comput. **41**(1), 42–52 (1997)
[2] Lenoski, D., Laudon, J., Gharachorloo, K., Weber, W.-D., Gupta, A., Hennessy, J.L., Horowitz, M., Lam, M.S.: The Stanford DASH multiprocessor. IEEE Comput. **25**(3), 63–79 (1992)
[3] Collet, J., Litaize, D., Campenhout, J.V., Desmulliez, M., Jesshope, C., Thienpont, H., Goodman, J., Louri, A.: Architectural approach to the role of optics in monoprocessor and multiprocessor machines. Appl. Opt. **39**(5), 671–682 (2000)
[4] Benner, A.F., Ignatowski, M., Kash, J.A., Kuchta, D.M., Ritter, M.B.: Exploitation of optical interconnects in future server architectures. IBM J. Res. Dev. **49**(4/5) 755–776 (2005)

[5] Mohammed, E. et al.: Optical interconnect system integration for ultra-short-reach applications. Intel Technol. J. **8**(2), 115–127 (2004)
[6] Schares, L., et al.: Terabus—a waveguide-based parallel optica 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, pp. 369–372. The Institution of Electrical Engineers, Glasgow, Scotland (2005)
[7] Heirman, W., Artundo, I., Desmet, L., Dambre, J., Debaes, C., Thienpont, H., Van Campenhout, J.: Speeding up multiprocessor machines with reconfigurable optical interconnects. In: Eldada, L., Lee, E.-H. (eds.) Proceedings of SPIE, Optoelectronic Integrated Circuits VIII, Photonics West, vol. 6124, p. 61240K. SPIE, San Jose, California, USA (2006)
[8] Katsinis, C.: Performance analysis of the simultaneous optical multi-processor exchange bus. Parallel Comput. **27**(8), 1079–1115 (2001)
[9] Heirman, W., Dambre, J., Van Campenhout, J.,Debaes, C., Thienpont, H.: Traffic temporal analysis for reconfigurable interconnects in shared-memory systems. In: Proceedings of the 19th IEEE International Parallel & Distributed Processing Symposium, IEEE Computer Society, p. 150. Denver, Colorado (2005)
[10] Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B.: Simics: A full system simulation platform. IEEE Comput. **35**(2), 50–58 (2002)
[11] Ridruejo, F., Gonzalez, A., Miguel-Alonso, J.: 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), pp. 547–553. Olso, Norway (2005)
[12] Brunfaut, M., Meeus, W., Van Campenhout, J., Annen, R., Zenklusen, P., Melchior, H., Bockstaele, R., Vanwassenhove, L., Hall, J., Wittman, B., Nayer, A., Heremans, P., Van Koetsem, J., King, R., Thienpont, H., Baets, R.: 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, pp. 160–171. Bellingham (2001).
[13] Huang, D., Sze, T., Landin, A., Lytel, R., Davidson, H.: Optical interconnects: out of the box forever?. IEEE J. Select. Top. Quant. Electron. **9**(2), 614–623 (2003)
[14] Snyder, L.: Introduction to the configurable, highly parallel computer. Computer **15**(1), 47–56 (1982)
[15] Pinkston, T.M., Goodman, J.W.: Design of an optical reconfigurable shared-bus-hypercube interconnect. Appl. Opt. **33**(8), 1434–1443 (1994)
[16] Han X., Chen R.T. (2004) Improvement of multiprocessing performance by using optical centralized shared bus. In: Proceedings of the SPIE, vol. 5358, pp. 80–89 (2004)
[17] Shacham, A., Small, B.A., Liboiron-Ladouceur, O., Bergman, K.: A fully implemented 12 × 12 data vortex optical packet switching interconnection network. J. Lightw. Technol. **23**(10), 3066–3075 (2005)
[18] Artundo, I., Desmet, L., Heirman, W., Debaes, C., Dambre, J., Van Campenhout, J., Thienpont, H.: Selective optical broadcast component for reconfigurable multiprocessor interconnects. IEEE J. Select. Topic. Quant. Electron.: Spec. Issue Opt. Commun. **12**(4), 828–837 (2006)
[19] Heirman, W., Dambre, J., Artundo, I., Debaes, C., Thienpont, H., Stroobandt, D., Van Campenhout, J.: Predicting reconfigurable interconnect performance in distributed shared-memory systems. Integ. the VLSI J. **40**(4), 382–393 (2007)
[20] Sterling, T., Savarese, D., Becker, D.J., Dorband, J.E., Ranawake, U.A., Packer, C.V.: Beowulf: A parallel workstation for scientic computation, In: Proceedings of the International Conference on Parallel Processing, pp. 11–14. CRC Press, Boca Raton, USA (1995)
[21] Sánchez, J.L., Duato, J., García, J. M.: Using channel pipelining in reconfigurable interconnection networks. In: 6th Euromicro Workshop on Parallel and Distributed Processing, 1998.
[22] Leiserson, C.E., Abuhamdeh, Z.S., Douglas, D.C., Feynman, C.R., Ganmukhi, M.N., Hill, J.V., Hillis, W.D., Kuszmaul, B.C., Pierre, M.A.S., Wells, D.S., Wong-Chan, M.C., Yang, S.-W., Zak, R.: The network architecture of the Connection Machine CM-5. J. Parallel Distributed Comput. **33**(2), 145–158 (1996)
[23] Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: Characterization and methodological considerations. In: Proceedings of the 22th International Symposium on Computer Architecture, pp. 24–36. Santa Margherita Ligure, Italy (1995).

Author Biographies

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 optical 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.

Inigo 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 such as the European Network of Excellence on Microoptics "NEMO". 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.