Communications Latency Hiding Techniques for a Reconfigurable Optical Interconnect: Benchmark Studies[1]

Ahmad Afsahi Nikitas J. Dimopoulos
Department of Electrical and Computer Engineering, University of Victoria
P.O. Box 3055, Victoria, B.C., Canada, V8W 3P6
{aafsahi, nikitas}@ece.uvic.ca

Abstract. Communication overhead adversely affects the performance of multicomputers. In this work, we present evidence (through the analysis of several parallel benchmarks) that there exists communications locality, and that it is "structured". We have used this in a number of heuristics that "predict" the target of subsequent communications. This technique, can be applied directly to reconfigurable interconnects (optical or conventional) to hide the communications latency by reconfiguring the interconnect concurrently to the computation.

1.0 Introduction

Message-passing multicomputers are composed of a large number of processor/memory modules that communicate with each other by exchanging messages through their interconnection networks. Optics is ideally suited for implementing interconnection networks because of its superior characteristics over electronics [13,15]; optical signals do not interact with each other, and an optical network can be reconfigured on demand. We have introduced [1] a *reconfigurable optical network*, OK_N, consisting of N computing nodes. A node is capable of connecting directly to any other node. Connections are established dynamically by reconfiguring the interconnect, and remain established until they are explicitly destroyed. A block diagram of the network is shown in Figure 1. Circuit-switching with k-port or *single*-port models with full-duplex communication is assumed.

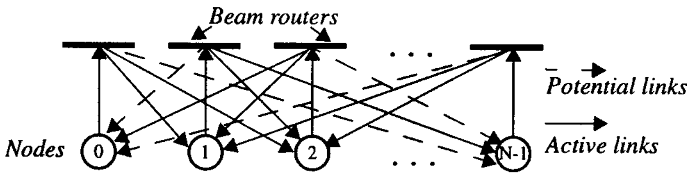


FIGURE 1. OKN, a massively parallel computer interconnected by a complete free-space optical interconnection network

The node-to-node communication delay is modeled as $T = d + t_s + l_m\tau$ with d being the reconfiguration delay, t_s the setup time, l_m the length of the message, τ the per unit transmission time. The setup time t_s [6] and reconfiguration delay d, are the major contributors of the communication delay T, both being of the order of several μs. Several researchers are working to minimize this communication delay by using *active* [7] or *fast* messages [14]. In this work, we are particularly interested in techniques that hide the reconfiguration delay, d.

It is obvious that if a link is already in place, then the configuration phase is not needed with a commensurate savings in the message transmission time. This can be

1. This research was supported through grants from the Natural Sciences and Engineering Research Council of Canada.

accomplished, if the target of the communication operation can be "predicted" before the message itself is available. If the communication operation is regular and known, it is possible to determine the destinations and the instances that these are to be used [1]. However, if the algorithm is not known, the above approach cannot be used.

Many parallel algorithms are built from loops consisting of computation and communication phases. Hence, communication patterns may be repetitive. This has motivated researchers to find *communications locality* properties for parallel applications [9,10]. By communications locality, we mean that if a certain source-destination pair has been used it will be reused with high probability by a portion of code that is "near" the place that was used earlier, and that it will be reused in the near future. If communications locality exists in parallel applications, then it is possible to *cache* the configuration that a previous communication request has made and reuse it at a later stage. Caching in the context of this discussion will mean that when a communication channel is established it will remain established until it is explicitly destroyed.

The main focus of our work is to explore whether the target of a communication request can be "predicted". For these studies, we used the MPI [11] implementation of five parallel applications (BT, SP, LU, MG, CG) of the NAS parallel benchmark suite (NPB) [3] on a network of SUN, and HP workstations using MPICH [8]. The target prediction heuristics developed adapt based on the past history and can be used in circuit switched interconnects including [5,16]. Section 2 analyzes the proposed heuristics, while we conclude with Section 3.

2.0 Latency hiding heuristics

The heuristics proposed in this section predict the destination node of a subsequent communication request based on a past history of communication patterns. Although our heuristics are applicable to any-port model, we shall present most of our results under the single port communications model. We use the *hit ratio* to establish and compare the performance of these heuristics. As a hit ratio, we define the percentage of times that the predicted destination node was correct out of all communication requests.

2.1 The LRU, FIFO, and LFU heuristics

The *Least Recently Used* (LRU) [9], *First-In-First-Out* (FIFO) and *Least Frequently Used* (LFU) heuristics, all maintain a set of k (k is the *window size*) message destinations. If the next message destination is already in the set, then a hit is recorded. Otherwise, a miss is recorded and the new destination replaces one of the destinations in the set according to which of the LRU, FIFO or LFU strategies is adopted.

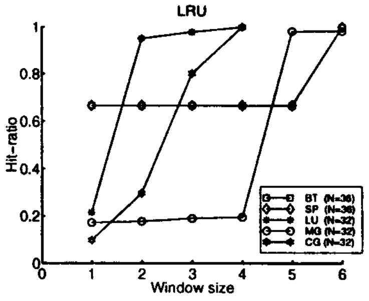


FIGURE 2. Effects of the LRU heuristic on the NAS benchmarks

The window size, k, corresponds to the number of ports used. Figure 2, shows the result of the LRU heuristic on the benchmarks. It can be seen that the hit-ratios in all benchmarks approach 1 as the window size increases. The performance of the FIFO algorithm is almost the same with LRU for all benchmarks. Additionally, the performance of the LFU algorithm [2] is better than LRU and FIFO, the exception being the LU benchmarks for $k = 2$.

2.2 The Single-cycle heuristic

The *Single-cycle* heuristic is based on the fact that if a group of destinations are requested cyclically, then a single port can accommodate these requests by ensuring that the connection to the subsequent node in the cycle can be established as soon as the current request ends. This heuristic implements a simple cycle discovery algorithm. Starting with a *cycle-head* node (this is the first node that is requested at start-up, or the node that causes a miss), we log the sequence of requests until the cycle-head node is requested again. This stored sequence constitutes a cycle and is used to predict the subsequent requests. If the predicted node coincides with the subsequent requested node, then we record a hit. Otherwise, we record a miss and the cycle formation stage commences with the cycle-head being the node that caused the miss.

The example to the right illustrates the heuristic used. The top trace represents the sequence of requested destination nodes, while the bottom trace represents the predicted nodes according to the Single-cycle heuristic. The arrows with the cross represent misses, while the ones with the circle represent hits. The "dash" in place of a predicted node indicates that a cycle is being formed; thus no prediction is offered.

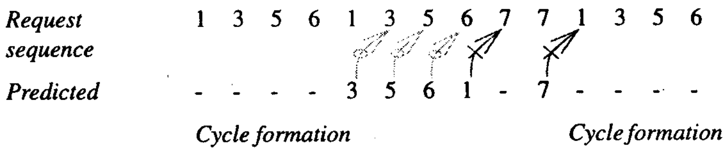
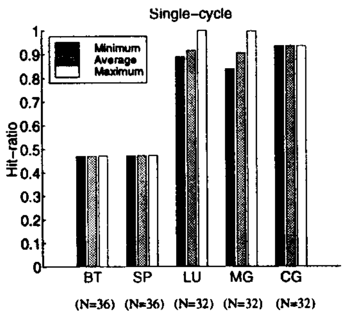


Figure 3, shows the behavior of this algorithm. This algorithm behaves much better than the LRU, FIFO and LFU heuristics for the LU, MG, and CG benchmarks. However its performance deteriorates for the BT and SP benchmarks. One of the reasons is that in these benchmarks there exist cycles of length one (such as the one composed of node 7 in the example above) always resulting into two misses. The Single-cycle2 heuristic presented next improves the performance ..



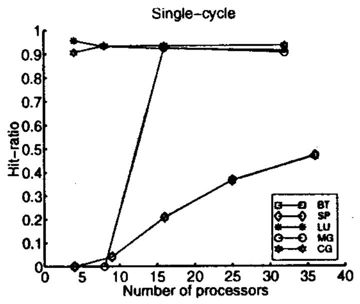


FIGURE 3. Effects of the Single-cycle heuristic on the NAS benchmarks

2.3 The Single-cycle2 heuristic

The *Single-cycle2* heuristic is identical to the single-cycle heuristic with the addition that during cycle formation, the previously requested node is offered as the predicted node. This heuristic performs much better than the LRU, FIFO, and LFU algorithms for the LU, MG, and CG benchmarks under single-port modeling and almost identically for the BT, and SP benchmarks [2].

2.4 The Tagging heuristic

The *Tagging* heuristic assumes a static communications environment in the sense that a particular communications request (send) in a section of code, will be to the same target node with a large probability. Therefore, as the execution trace nears the section of code in question, it can cause the communications environment to establish the connection to the target node before the actual communications request is issued[1].

For our experiments, we attach a different tag to each of the communication requests found in the benchmarks. To this tag, we assign the requested target node. A hit is recorded if in subsequent encounters of the tag, the requested communications node is the same as the target already associated with the tag. Otherwise, a miss is recorded and the tag is assigned the newly requested target node. This tagging technique is similar to the technique used in the branch cache of the MC68060 [4].

As it can be seen in Figure 5 the tagging heuristic results in an excellent performance (hit ratios in the upper 90%) for all the benchmarks except CG. The reason is that the CG benchmark includes send operations with a target address calculated based on a loop variable. Thus, the same section of code cycles through a number of different target addresses. As we have seen before, the Single-cycle and the Single-cycle2 heuristics are excellent in discovering such cyclic occurrences. In the following, we propose combined tag and cycle heuristics which are able to alleviate this problem.

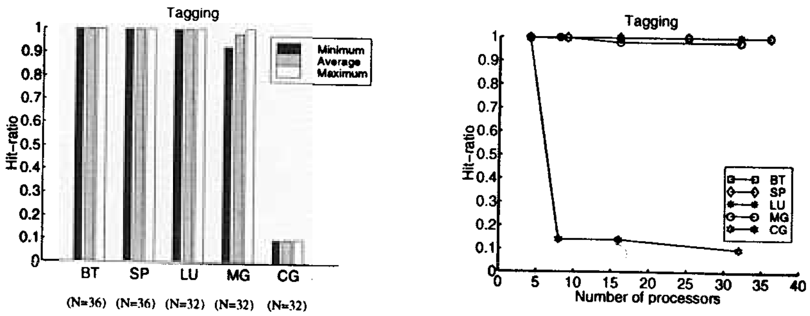


FIGURE 4. Effects of the Tagging heuristic on the NAS benchmarks

2.5 The Tag-cycle and Tag-cycle2 heuristics

In the *Tag-cycle* heuristic, we attach a different tag to each of the communication requests found in the benchmarks and do a Single-cycle discovery algorithm on each tag. To each tag, we assign the sequence of requested target nodes. We use these sequences in the same manner as in the Single-cycle and Single-cyle2 heuristics (c.f.

1. This can be implemented with the help of the compiler through a *pre-connect(tag)* operation which will force the communications system to establish the communications connection before the actual communications request is issued. This technique is similar to the prefetch operation advocated by T. Mowry and A. Gupta[12].

Section 2.2 and Section 2.3). The Tag-cycle heuristic performs exceptionally well across all the benchmarks, as shown in Figure 5.

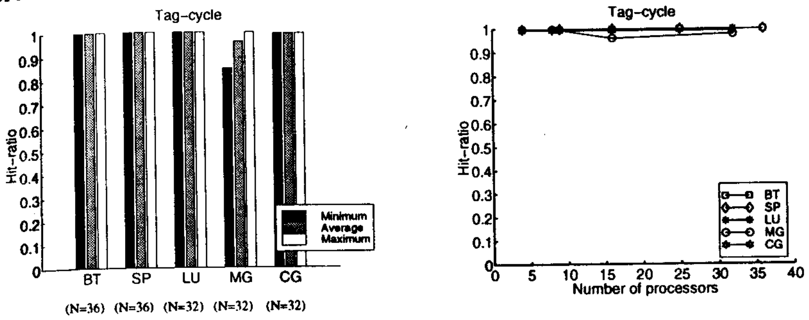


FIGURE 5. Effects of the Tag-cycle heuristic on the NAS benchmarks

The *Tag-cycle2* heuristic is identical to the Tag-cycle heuristic with the addition that during cycle formation, similar to the Single-cycle2 heuristic, the previously requested node is offered as the predicted node. The performance of Tag-cycle2 heuristic is slightly better than Tag-cycle.

3.0 Discussion and Conclusions

In this work, we presented a number of heuristics which can be used to "predict" the target of a communications request before the actual request is issued. These heuristics use the pattern of communications and are designed to extract dependencies which are embedded in these patterns. For these studies, we used the publicly available NAS parallel benchmarks. The results of our studies give strong evidence for the existence of "communications locality" at least in the benchmark programs studied.

The heuristics proposed are only possible because of the existence of *communications locality*. This is a very desirable property since it allows us to effectively hide the cost of establishing such communications links, providing thus the application with the raw power of the underlying hardware (e.g. a reconfigurable optical interconnect).

Figure 6, presents a comparison of the performance of the heuristics presented in this work only under single-port assumption. That is only one communications channel is available at any time, and this is reconfigured on demand. As it can be seen, the tagging+cycle2 heuristic is the best for all benchmarks and its hit-ratio is consistently very high (approaching 100%) for all the benchmarks considered here.

We are confident that the existence of "communications locality" and the resulting latency hiding techniques will usher a new era in interconnection technologies by allowing the use of reconfiguarbility and fast optical fabrics.

References

1. A. Afsahi and N. J. Dimopoulos, "Collective Communications on a Reconfigurable Optical Interconnect," Proceedings of the International Conference on Principles of Distributed Systems, Dec., 1997, pp. 167-181
2. A. Afsahi and N. J. Dimopoulos "Communications Latency Hidinng Techniques for a Reconfigurable Optical Interconnect: Benchmark Studies" *Technical Report* ECE-98-2, Department of Electrical and Computer Engineering, University of Victoria, June 1998.

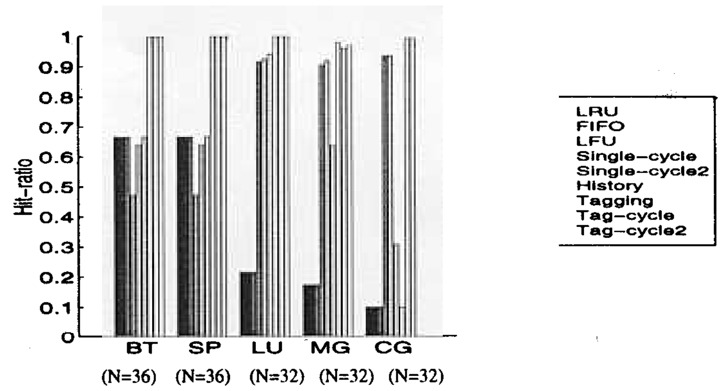


FIGURE 6. Comparison of the performance of the heuristics discussed in this work for the five NAS benchmarks under single-port modelling

3. D. H. Bailey, et al., "NAS Parallel Benchmark Result 3-94," Proceedings of Scalable High-Performance Computing Conference, 1994, pp. 111- 120
4. J. Circello, et al., "The Superscalar Architecture of the MC68060," IEEE Micro, Volume 15, Number 2, April 1995, pp. 10-21
5. B. V. Dao, S. Yalamanchili, and J. Duato, "Architectural Support for Reducing Communication Overhead in Multiprocessor Interconnection Networks" Proceedings, Third International Symposium on High Performance Computer Architecture, 1997, pp. 343-352
6. J. J. Dongarra and T. Dunigan, "Message-Passing Performance of Various Computers," Concurrency, Vol. 9, No. 10, Dec. 1997, pp. 915-926
7. T. V. Eicken, et al., "Active Messages: A Mechanism for Integrated Communication and Computation," Proceedings of the 19th Annual International Symposium on Computer Architecture, May 1992, pp. 256-265
8. W. Gropp and E. Lusk, "User's Guide for MPICH, a Portable Implementation of MPI," Argonne National Laboratory, Mathematics and Computer Science Division, ANL/MCS-TM-ANL-96/6
9. J. Kim and D. J. Lilja, "Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs, "Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, International Symposium on High Performance Computer Architecture, February 1998, pp. 202-216
10. D. G. de Lahaut and C. Germain, "Static Communications in Parallel Scientific Programs" Proceedings of PARLE'94, Parallel Architecture and Languages, Athen, Greece, July 1994
11. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. Version 1.1 (June 1995)
12. T. Mowry and A. Gupta, "Tolerating Latency Through Software-Controlled Prefetching in Shared-Memory Multiprocessors," Journal of Parallel and Distributed Computing, 12(2), 1991, pp. 87-106
13. R . A. Nordin, et al., "A System Perspective on Digital Interconnection Technology," IEEE Journal of Lightwave Technology, Vol. 10, June 1992, pp. 801-827
14. S. Pakin, M. Lauria, and A. Chien, " High Performance Messaging on Workstation: Illinois Fast Messages (FM) for Myrinet," Proceedings of Supercomputing'95, Nov., 1995
15.G. I. Yayla, P.J. Marchand and S.C. Esener "Speed and Energy Analysis of Digital Interconnections: Comparison of On-chip, Off-chip and Free-Space Technologies" Applied Optics, Vol. 37, No.2, Jan. 1988, pp. 205-227.
16. X. Yuan, R. Melhem, and R. Gupta, "Compiled Communication for All-Optical TDM Networks," Proceeding's of Supercomputing'96, 1996