Chapter 14

COMMUNICATION LATENCY HIDING IN RECONFIGURABLE MESSAGE-PASSING ENVIRONMENTS: QUANTITATIVE STUDIES

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

Abstract Communications overhead is one of the most important factors affecting performance in message-passing multicomputers. We present evidence that there exists communications locality, and that this locality is "structured". We propose a number of heuristics that can be used to "predict" the target of subsequent communication requests. Communication latency is hidden through reconfiguring the network concurrently to the computation. Quantitative results obtained from standard parallel benchmarks run on IBM SP systems are also presented.

Keywords: Message-Passing, Latency-Hiding, Communication Locality, MPI, Reconfigurable Interconnects.

1. INTRODUCTION

Message-passing multicomputers are composed of a number of computing nodes that communicate with each other by exchanging messages through their intercon- nection networks. Optics is ideally suited for implementing interconnection net- works because of its superior characteristics over electronics (Yayla et al., 1998; Nordin et al., 1992). Various optical interconnection networks including the works in (Bourdin et al., 1995; Louri et al., 1994) have been proposed.

In free-space optical interconnects, optical signals can propagate very close to each other and pass each other without interaction and may reconfigure

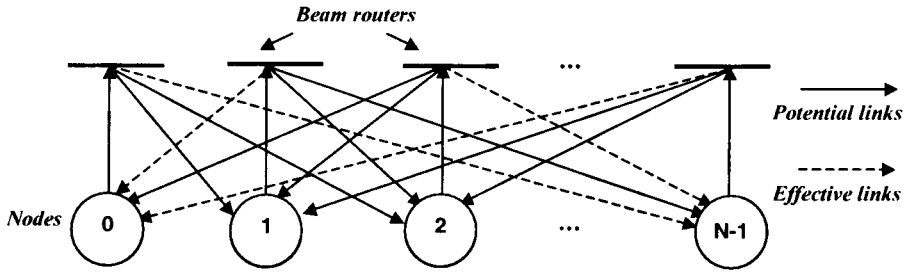


Figure 14.1 RON (k, N), a massively parallel computer interconnected by a complete free-space optical interconnection network.

on demand (Marchand et al., 1997). Free-space optical interconnects use space (vacuum, air or glass) for optical signal propagation and include the optoelectronic device(s) for photon generation or modulation, and the optical beam router to redirect or distribute optical beams.

Definition: A *reconfigurable optical network, RON (k, N)*, (Afsahi et al., 1997) consists of N computing nodes with their own local memory. A node is capable of connecting directly to any other node. A node can establish k simultaneous connections. These connections are established dynamically by reconfiguring the optical interconnect. The links remain established until they are explicitly destroyed.

A simplified block diagram of the network is shown in Figure 14.1. Messages are sent using circuit-switching. Each node can simultaneously send and receive k messages on its k links, one message on each link, (the *k-port* model), or exactly one message on one of its links (the *single-port* model). Full-duplex communication where a node can send and receive messages at the same time is supported.

Various implementation technologies exist to embody the above abstract model, including computer generated holograms and deformable mirrors for switching, frequency hoping for coding, wavelength tuning for transceivers, VCSELs and SEEDs for photon generation or modulation We assume that one or more of the technologies outlined above will be used to implement such an interconnect. Under such an implementation, the various overheads associated with the reconfiguration of the network are lumped together as the reconfiguration delay d.

The communication time T required to send a message from one node to another. In the linear model, the communication time depends, among other things, on the length of the message, and it is formulated as $T = t_s + l_m\tau$ where l_m is the length of the message, τ is the per unit transmission time, and t_s, the setup time, is the time required to prepare the message, such as adding a header, a trailer, memory copying etc. For our case, we amend the linear model by explicitly including the reconfiguration delay d that is necessary for

a node to configure a link that would connect directly to its target node. The transmission time then becomes $T = d + t_s + l_m\tau$. The time on the fly, $l_m\tau$, is negligible compared to the setup time, t_s, and the reconfiguration delay, d. In the current generation of parallel computer systems, the setup time is several 10s of microseconds (Dongarra et al., 1997). Several researchers are working to minimize this cost by user-level messaging techniques such as active messages (Eicken et al., 1992) and fast messages (Pakin et al., 1995). In this work we are interested in techniques that hide the reconfiguration delay, d.

We shall assume that a node sends a message to another node by first establishing a link to the target (hence the reconfiguration delay d) and then sending the actual message over the established link. It is obvious that if the link is already in place, then the configuration phase does not enter the picture with a commensurate savings in the message transmission time. The main objective of this work is therefore to establish efficient algorithms where the link establishment costs are minimized. The stated objective can be accomplished, if the target of the communication operation can be "predicted" before the message itself is available.

If the communication pattern is regular, then it is possible to determine the destinations and the instances that these shall be used. We have developed such algorithms for broadcasting (Afsahi et al., 1997). However, if the communications pattern is not known, regular or simple, the above approach cannot be used.

In the context of the shared memory programming, there are several works on hardware-controlled and software-controlled prefetching of the next shared data request (Mowry et al., 1991; Sakr et al., 1997; Zhang et al., 1995). T. Mowry and A. Gupta (Mowry et al., 1991) have used prefetching, multithreading, and caching to hide/reduce the latency in shared memory multiprocessors. M. F. Sakr et al. (Sakr et al., 1997) have used time series and neural networks for the prediction of the next memory requests in shared memory multiprocessors.

In the context of message passing programming, many parallel algorithms are built from loops that include computation and communication phases. This has motivated researchers to find communications locality properties of parallel applications (Kim et al., 1998; Lahaut et al., 1994). J. Kim and D. J. Lilja (Kim et al., 1998) have recently shown that there is locality in message destination, message sizes, and consecutive runs of send/receive primitives in parallel algorithms.

In conjunction with this work, *communications locality* will mean that if a certain source-destination pair has been used, it will be re-used with high probability by a portion of code that is "near" the place that was used earlier, and that it will be re-used 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.

This paper is divided into two parts. The first part (Afsahi et al., 1999a; Afsahi et al., 1999b) explores the ability of a number of heuristics to predict the target of a communication request. For these studies, we have utilized a number of parallel benchmarks, and extracted the communication traces on which we applied our heuristics. These heuristics can be used either dynamically in real-time at the communication assist/network interface and/or off-line during the compilation phase. The proposed heuristics can be used in circuit-switched network including the wave switching (Dao et al., 1997) and (Yuan et al., 1996).

The second part considers the execution time of the computation phases of these parallel benchmarks on an IBM SP2 system and examines whether the execution times are sufficiently large for the reconfigurations to proceed concurrently with the computation. We also present the improvement on the total reconfiguration delay attained through the use of the said target prediction heuristics.

In the following section, we shall discuss the benchmarks studied in this paper. Section 3. explains the proposed target prediction heuristics and presents their performance in terms of the parallel benchmarks. In section 4., we compare the inter-send computation times of the benchmarks on the IBM SP2 system with different reconfiguration costs, and present the performance enhancements of the proposed heuristics on the total reconfiguration times, in section 5.. The heuristics' effects on the receiving side are in section 6.. and we conclude with section 7..

2. PARALLEL BENCHMARKS

We have used some well-known parallel benchmarks from the NAS parallel benchmarks suite (NPB) (Bailey et al., 1994), the Parallel Spectral Transform Shallow Water Model (PSTSWM) (Worley et al., 1994), and the pure QCD Monte Carlo Simulation Code with MPI (QCDMPI) (Hioki 1996). The NAS parallel Benchmarks (NPB) is a set of eight benchmark problems, each of which focuses on some aspects of highly parallel supercomputing for aerophysics applications. The NPB consists of five "kernels", and three "simulated computational fluid dynamic applications". In this paper, we are only interested in the patterns of the point-to- point communications. Therefore the EP, FT, and IS kernels are not suitable for our study. EP and FT use only collective communication operations while each node in the IS kernel always communicates with a specific node.

PSTSWM is a message-passing benchmark code and parallel algorithm testbed that solves the nonlinear shallow water equations on a rotating sphere using the spectral transform method.

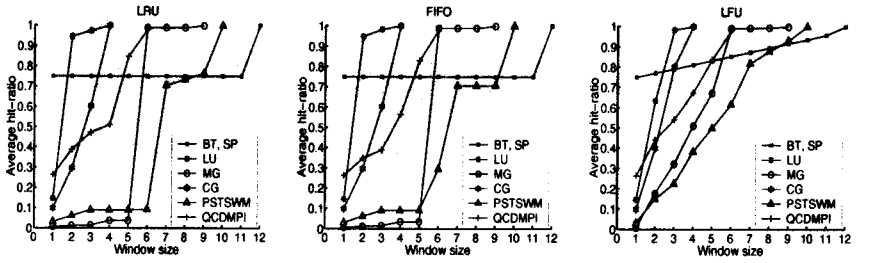


Figure 14.2 LRU, FIFO, and LFU heuristics' effects on the benchmarks, when N = 64.

QCDMPI is a pure Quantum Chromo Dynamics simulation code with MPI calls. It is a powerful tool to analyze the non-perturbative aspects of QCD. and can be applied to any dimensional QCD.

We have used the MPI (Message Passing Interface Forum, 1995) implementation of the NAS benchmarks (version 2.3, W and A classes), the PSTSWM (version 6.2), and the QCDMPI (version 1.4) benchmarks on the IBM SP2. We wrote our own profiling codes using the wrapper facility of the MPI to gather the communication traces and the timing profiles of these applications.

3. LATENCY HIDING HEURISTICS

The set of heuristics proposed in this section predict the destination node of a subse- quent communication request based on a past history of communication patterns on a per source node basis. These heuristics can be categorized into three different sets: the classical LRU/FIFO/LFU heuristics, the Cycle-based heuristics, and the Tag- based heuristics (Afsahi et al., 1999a; Afsahi et al., 1999b).

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.

3.1 LRU, FIFO AND LFU HEURISTICS

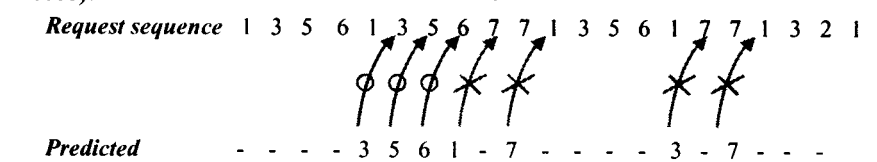
The *Least Recently Used* (LRU), *First-In-First-Out* (FIFO) and *Least Frequently Used* (LFU) heuristics, all maintain a set of k (k is the *window size*) message destinations (Afsahi et al., 1999b). If the next message destination is already in the set, then a hit is recorded. Otherwise, a miss is 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. The window size, k, corresponds to the number of ports used. Figure 14.2, shows the results of the LRU, FIFO, and LFU heuristics on the benchmarks when the number of processors is 64. It is clear that the hit-ratios in all benchmarks approach 1 as the window size increases.

The performance of the FIFO algorithm is almost the same as the LRU for all benchmarks. However, the LFU algorithm has a better performance than the LRU and FIFO heuristics, the exception is for the LU benchmark, when $k =$ and $=$, 32, and 64 (Afsahi et al., 1999b).

3.2 CYCLE HEURISTICS

The LRU, FIFO, and the LFU heuristics perform better when k is sufficiently large. However, this adds to the hardware complexity, as k links should be setup before the next message is ready to take place. Therefore, we consider other heuristics that perform well under single-port modeling.

Cycle heuristics are based on the fact that if a group of destinations are requested repeatedly in a cyclical fashion, then a single port can accommodate these requests by ensuring that the connection to the subsequent node in the cycle can be estab- lished as soon as the current request terminates. These heuristics implement 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 can be used to predict the subsequent requests. If the predicted node coincides with the subsequent requested node, then we record a hit. If the requested node does not coincide with the predicted one, then we record a miss and the cycle formation stage commences with the cycle-head being the node that caused the miss. The following example illustrates the heuristic used. The top trace represents the sequence of requested destination nodes, while the bottom trace represents the predicted nodes set 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, and therefore no predicted nodes are offered (note that this is also added to the misses).



The enhancements to the Single-cycle heuristic were introduced in the *Single- cycle2* (Afsahi et al., 1999a; Afsahi et al., 1999b) where during cycle formation, the previously requested node is offered as the predicted node.

3.3 BETTER-CYCLE AND BETTER-CYCLE2 HEURISTICS

In the Better-cycle heuristics (*Better-cycle* and *Better-cycle2* (Afsahi et al., 1999a; Afsahi et al., 1999b), we maintain the previously formed cycle and upon

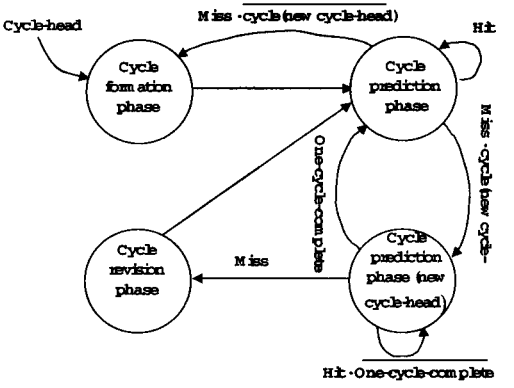


Figure 14.3 State diagram of the Better-cycle algorithm.

a miss, we attempt to use it. In case that the previous cycle is not suitable, a brand new cycle formation phase is entered. The state diagram used for the Better-cycle heuristic is given in Figure 14.3.

The *Better-cycle2* heuristic is identical to the Better-cycle heuristic with the addi- tion that during cycle formation and cycle revision phases the previously requested node is offered as the predicted node. As was expected, the Better-cycle2 heuristic performs the best among the cycle-based heuristics, and much better than the LRU, LFU, and FIFO heuristics. Figure 14.4 presents the average hit ratios of this heuristic.

3.4 TAGGING HEURISTIC

This heuristic assumes that the execution trace visits a particular communication request (e.g. *mpi_send*) several times and that the target node of this communication request remains the same with high probability. Therefore, as the execution trace nears the section of code in question, it can cause the communication environment to establish the connection to the target node before the actual communications request is issued. This can be implemented with the help of the compiler or by the programmer through a *pre-connect (tag)* operation (this is similar to (Mowry et al., 1991) which will force the communication system to establish the connection before the actual communication request is issued.

A different tag is attached to each of the communication requests found in the benchmarks. To this tag at the communication assist, we assign the requested target node. A hit is recorded if in subsequent encounters of the tag, the requested commu- nication 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. The performance of the Tagging heuristic is presented in Figure 14.5. The Tagging heuristic results in an excellent performance (hit

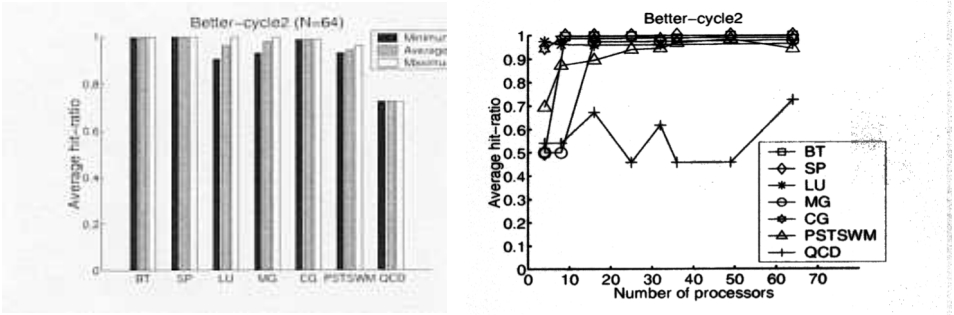


Figure 14.4 Effects of the Better_cycle2 heuristic on the benchmarks.

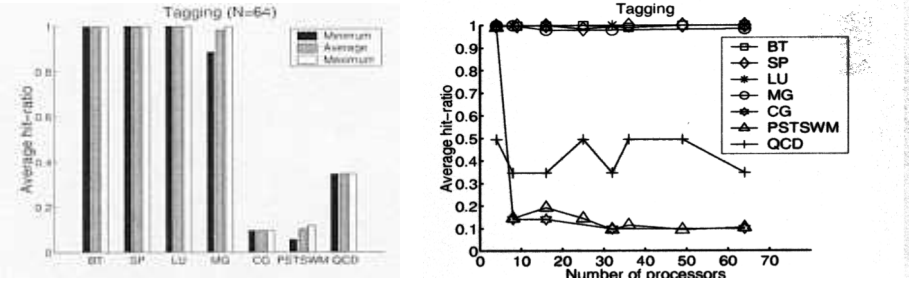


Figure 14.5 Effects of the Tagging heuristic on the benchmarks.

ratios in the upper 90%) for all the benchmarks except the CG, PSTSWM, and the QCDMPI benchmarks. The reason is that these benchmarks include send operations with a target address calculated based on loop variables. Thus, the same section of code cycles through a number of different target addresses. As we have seen in section 3.3, the Better-cycle and Better-cycle2 heuristics are excellent in discovering such cyclic occurrences for the CG and PSTSWM benchmarks. Meanwhile, the Better-cycle2 heuristic has better performance for the QCDMPI compared to the Tagging and other cycle heuristics.

3.5 TAG-BETTERCYCLE AND TAG-BETTERCYCLE2 HEURISTICS

We combined the Better-cycle and the tagging heuristics for better performance. In the *Tag-bettercycle* heuristic, we attach a different tag to each communication request found in the benchmarks and do a Better-cycle discovery algorithm on each tag. The *Tag-bettercycle2* heuristic is identical to the Tag-bettercycle heuristic with the addition that during cycle formation, as in Better-cycle2, the previously requested node is offered as the predicted node. The performance of this heuristic is shown in Figure 14.6. It is clear that its performance is superior to all other heuristics.

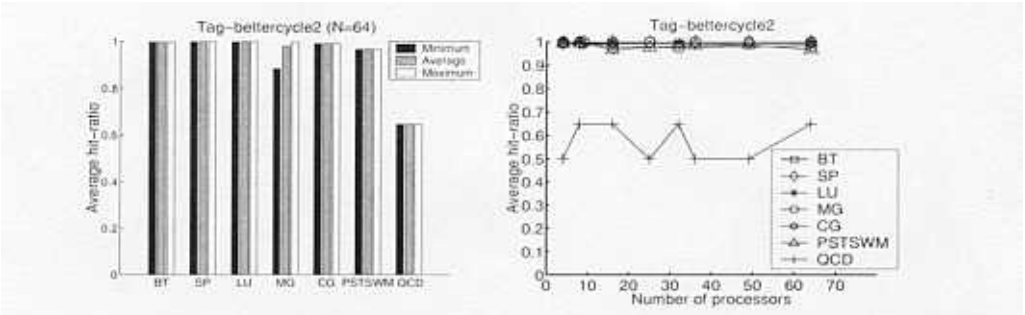


Figure 14.6 Effects of the Tag-bettercycle2 heuristic on the benchmarks.

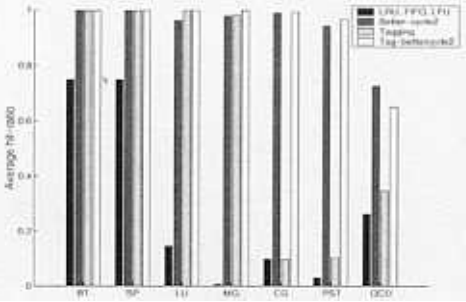


Figure 14.7 Comparison of the performance of the heuristics for the benchmarks under single-port modeling when number of processors is 64 (PST is PSTSWM).

Figure 14.7, presents a comparison of the performance of the heuristics presented in this work under single-port assumption when the number of processors is 64. That is a single communications channel is available to each node, and it is reconfigured on demand. The superior performance of the Tag-bettercycle2 heuristic is evident.

4. INTER-SEND COMPUTATION TIMES

To reconfigure the interconnect concurrently to the computation, two conditions are necessary: (1) An accurate prediction of the destination; (2) Enough lead time so that the reconfiguration be completed before the communication request arrives.

In the previous section, we presented a number of heuristics that can be used to predict the destination of the subsequent communication request. In this section, we shall argue that, at least in the benchmarks studied, there is sufficient computation time preceding a communication request to effectively hide the reconfiguration cost.

We have used the thirty node IBM SP2 Deep Blue machine at the IBM TJ Watson Research center and run the suite of benchmarks at the user space, one process per node, and under an exclusive access to the node. This avoided any

Table 14.1 Minimum inter-send computation times of the parallel benchmarks (all times in microseconds)

	4 nodes		8 nodes		9 nodes		16 nodes		25 nodes	
	W	A	W	A	W	A	W	A	W	A
BT	4.161	4.576	—		4.161	4.472	4.161	4.472	4.161	4.576
SP	4.161	4.784	—		4.161	4.472	4.161	4.472	4.161	4.368
LU	9.568	22.568	8.216	12.688	—		8.112	9.672	—	
MG	6.344	7.592	5.720	7.176	—		5.928	6.760	—	
CG	407.99	829.92	6.864	7.176	—		7.384	6.656	—	
PSTSWM	7.176		6.240		—		6.032		6.344	
QCD	1392.352		695.344		—		353.080		193.128	

task switching that might have affected our measurements. Our measurements determined a lower bound on the inter-send computation times (i.e. the time devoted to computation between two send requests). The inter-send computation measurements excluded any overhead associated with other communication primitives (e.g. receive) and it can thus be considered as a lower bound on the pure computation.

Table 1, shows the minimum inter-send computation times of the benchmarks on up to 25 nodes of the IBM Deep Blue machine. Examining the distribution of the inter-send times (Figure 14.8) revealed that they are distributed widely.

A variety of techniques are used to reduce the reconfiguration time in optical interconnects. In (Panajotov et al., 1998), the authors present a reconfiguration time of 25 s for an experimental reconfigurable optical interconnect. We compare the pure computation times of the benchmarks with this 25 s reconfiguration time, and with reconfiguration times of 10, 5, and 1 s as a measure of future advancements in this area. Figure 14.8 presents the percentage of the number of computation times more than 5, 10, and 25 s for each node of the SP2 and for each application. It is evident that the majority of the reconfigurations can proceed in parallel with the computation and be readied before the end of the computation.

5. TOTAL RECONFIGURATION TIME

In this section, we shall examine and quantify the effectiveness of the proposed heuristics. in hiding the reconfiguration delays.

We assume a multicomputer with nodes similar to the thin nodes of an IBM SP2 but with a reconfigurable optical interconnect which has a reconfiguration delay d (d=25, 10, 5, 1 s). The calculations which quantify the reconfiguration hiding capabilities of our heuristics, use the lower bound of the inter-send computation times. This allows us to compute the lower bound of the time that can be hidden. The algorithm used to obtain the time spent in reconfiguring the interconnect with and without the prediction heuristics is given by the following pseudocode.

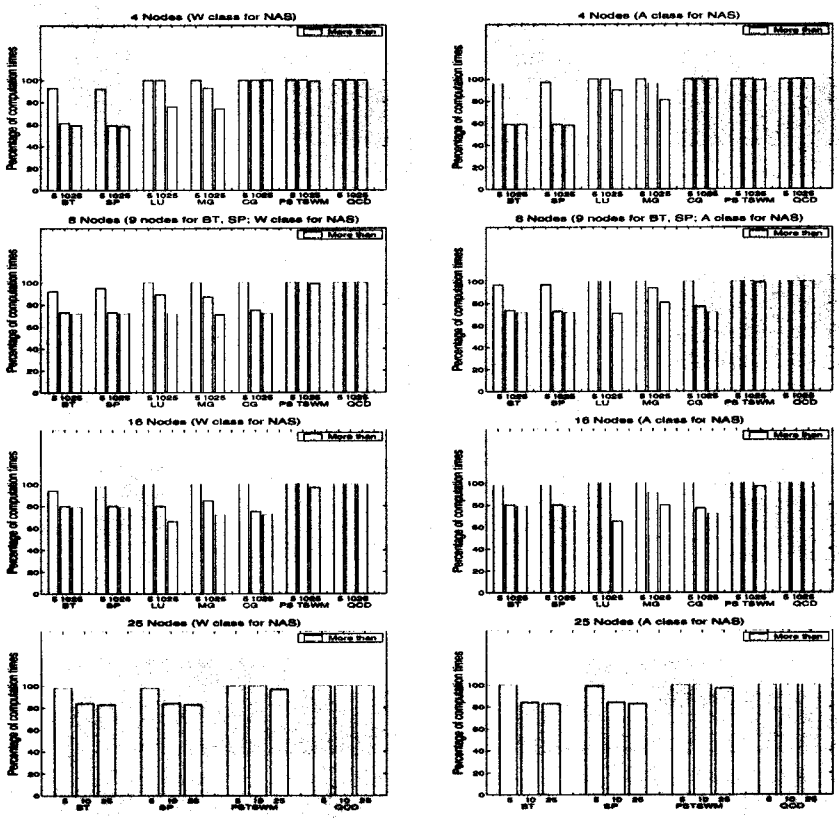


Figure 14.8 Percentage of the inter-send computation times for different benchmarks more than specific length of times when $N = 4, 8, 9, 16$, and 25

```
total_new_reconfiguration = 0.0;
total_original_reconfiguration = 0.0;
for each inter_send computation time {
  if (hit) then
    if (inter_send_computation < reconfiguration_delay) then
       total_new_reconfiguration += reconfiguration_delay - inter_send_computation;
  else total_new_reconfiguration += reconfiguration_delay;
  total_original_reconfiguration += reconfiguration_delay;
}
```

Figure 14.9, presents the total reconfiguration time obtained while employing heu- ristics as a percentage of the total configuration time when no prediction heuristics were employed when the number of processors is 16 (Afsahi et al., 1999b). The results are shown for the best heuristic of each category, that is the LRU/LFU/FIFO, Better-cycle2, and Tag-bettercycle2 heuristics.

The Benchmarks assumed a current generation and a 10 times faster CPU. The results are consistent with the fact that we can use most of the time to hide the reconfiguration delay with the help of one of the proposed high hit-ratio heuristics. Figure 14.10, shows a summary for the Tag-bettercycle2 heuristic up to 25 processors when the reconfiguration delay, d, is 25 microseconds.

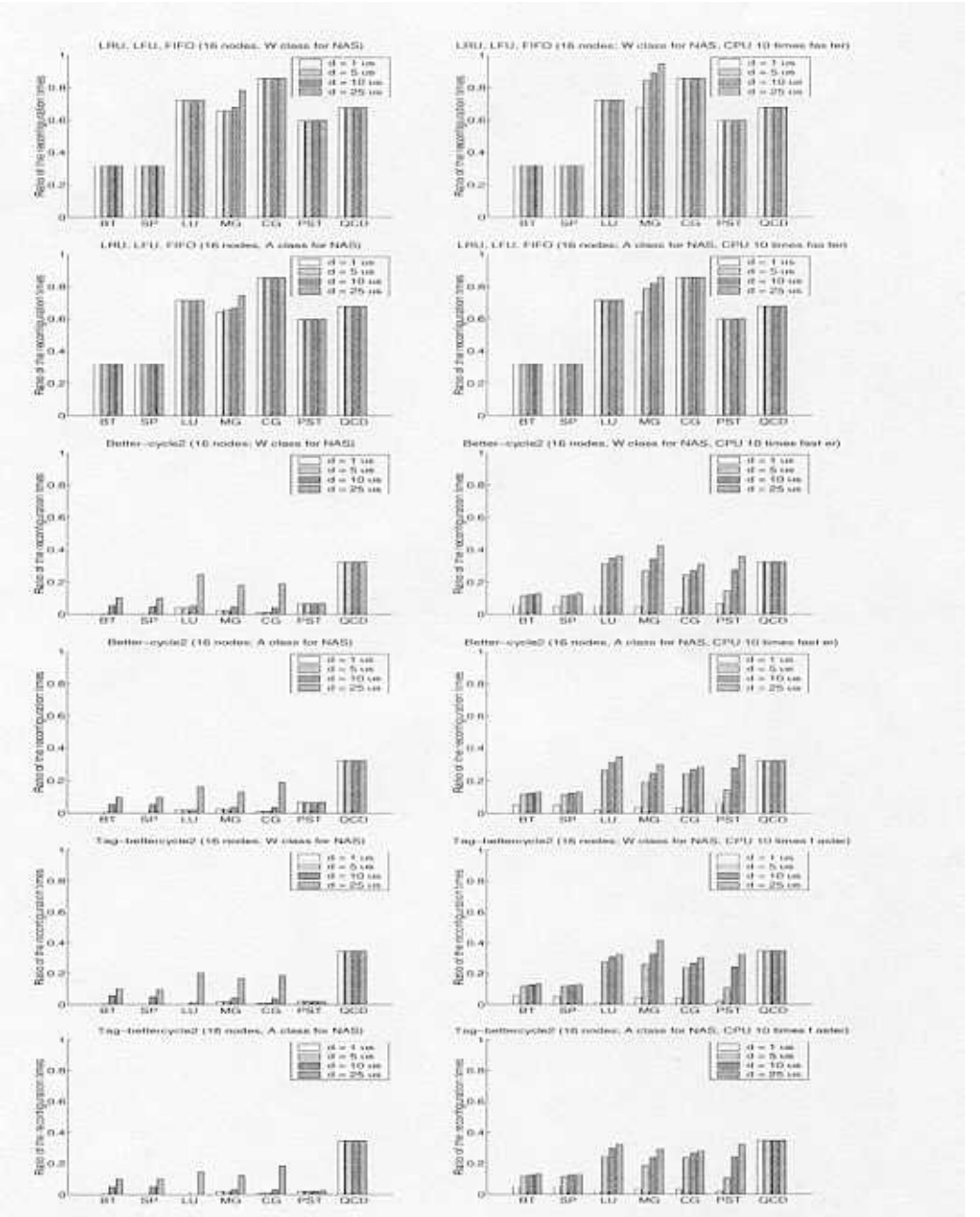


Figure 14.9 The total reconfiguration time obtained while employing the prediction heuristics as a percentage of the total configuration time when no prediction heuristics were employed. Benchmarks assumed current generation and a 10 times faster CPU with reconfiguration delay, = 1, 5, 10, and 25 s (PST is the PSTSWM).

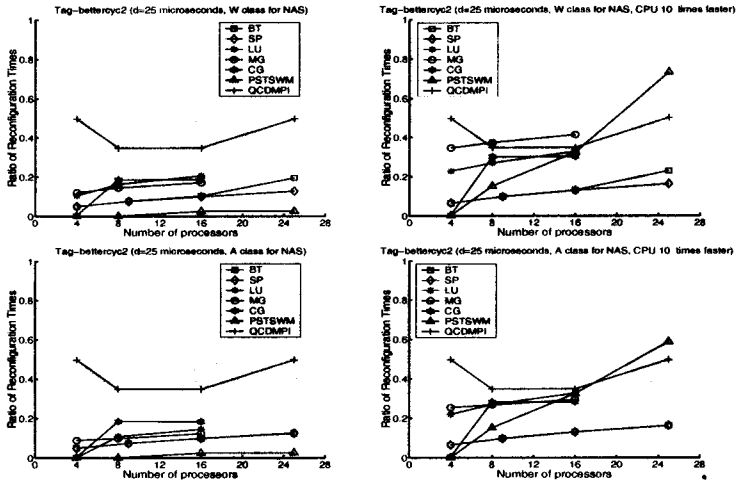


Figure 14.10 Summary of the ratio of the total reconfiguration times when applying the Tag-bettercycle2 heuristic on the benchmarks when the reconfiguration delay, d, is 25 microseconds.

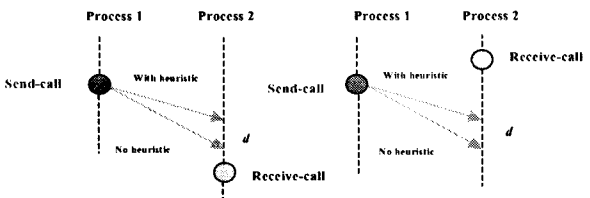


Figure 14.11 Heuristics' effects on the receiving side.

6. HEURISTICS' EFFECT ON THE RECEIVE SIDE

It is interesting to discover what would be the effect of the heuristics on the receiving sides. Using one of the high hit-ratio heuristics reduces the total reconfiguration delay. When these happen at the sender sides, most of the time the messages are delivered sooner at the receiver sides. If the receive calls have been issued after the corresponding send calls there would be no gain. However, if they are issued earlier then there would be performance enhancement on the receiving side and therefore on the whole execution time. This is shown in the Figure 14.11.

We present the average percentage of the times that the receive calls are issued earlier than their corresponding send calls for the CG, SP, and PSTSWM bench- marks, in Figure 14.12. For these, we synchronized the timing traces of each node of these applications. These ratios present the worst case scenario for the receive side improvements. We are currently working on deriving the performance enhancements of the heuristics (applied on the sender sides) for the receiver sides using an event-driven simulator.

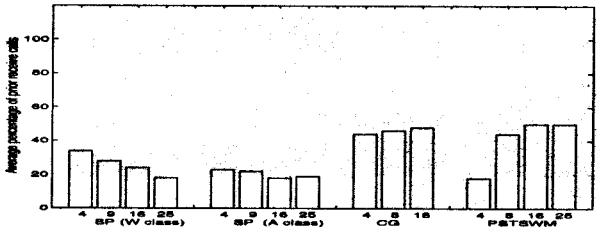


Figure 14.12 Average percentage of the times the receive calls are issued before the corresponding send calls when $N = 4, 8, 9, 16$, and 25.

7. CONCLUSION

In the first part of this work, we presented a number of heuristics that can be used to "predict" the target of a communication 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 NAS NPB suite, the PSTSWM and the QCDMPI parallel benchmarks. The results of our studies give strong evidence for the existence of *communications locality.*

The heuristics proposed are only possible because of the existence of communications locality that can be used in establishing a communication pathway between a source and a destination before this pathway is to be used. 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).

In the second part of our work, we presented the pure computation time of these parallel benchmarks on the IBM Deep Blue using its high performance switch and the user space when we had exclusive access to the nodes. In measuring the execution times of the computation phases we ensured that any system and communication overheads were excluded. In essence, the reported times are the lower bounds of the execution times of the computation times. We presented the percentage of times that the computation times are less than 1, 5, 10, and 25 microseconds. Even for current optical technology ($d =$) (Panajotov et al., 1998), the results show that we can use most of this time to hide the reconfiguration delay if we use one of the proposed high hit-ratio heuristics. We also presented the performance enhancements of the proposed heuristics on the total reconfiguration time. For this, we used the obtained computation/communi- cation traces and heuristics hit/miss profiles to determine the total reconfiguration time under different reconfigurable costs and processor speeds. The results indicated that the tag-cycle based heuristics have the best performance. We presented the average percentage of the times that the receive calls are issued earlier than their corresponding send calls. We are currently working on deriving the performance enhancements of the heuris-

tics (applied on the sender sides) for the receiver sides using an event-driven simulator. Finally, we are confident that the existence of "communications locality" and the resulting latency hiding techniques will usher a new era in interconnection technolo- gies by allowing the use of reconfiguarbility and fast optical fabrics.

Acknowledgments

This work was supported by grants from NSERC and the University of Victoria. We would like to thank Dr. Murray Campbell at the IBM TJ Watson Research Center and Mr. Greg Schick at the IBM Victoria for their kind help in accessing the IBM Deep Blue, and the staff of the computer center at the University of Victoria for the access to the university's IBM SP2 during the early stages of this work.

References

Afsahi, A. and Dimopoulos, N.J. (1997). Collective Communications on a Reconfigurable Optical Interconnect. *Proc. of the Int'l Conf. on Principles of Distributed Systems*, pp. 167-181.

Afsahi, A. and Dimopoulos, N.J. (1999a). Hiding Communication Latency in Reconfigurable Message-Passing Environments. *Proc. of the IPPS/SPDP 1999, 13th Int'l Parallel Processing Symp. and 10th Symp. on Parallel and Distributed Processing*, pp. 55-60.

Afsahi, A. and Dimopoulos, N.J. (1999b). Hiding Communication Latency in Reconfigurable Message-Passing Environments. Technical report ECE-99-3, *Dept. of Electrical and Computer Engineering*, Univ. of Victoria.

Bailey, D.H. Barszcz, E., Dagum, L. and Simon, H.D. (1994). NAS Parallel Benchmark Result 3-94. *Proc. of the Scalable High-Performance Comp. Conf.* , pp. 111- 120.

Bourdin H., Ferriera, A., and Marcus, K. (1995). A Comparative Study of One-to-Many WDM Lightwave Interconnection Networks for Multiprocessors. *Proc. of the 2nd Int'l Conf. on Massively Parallel Processing using Optical Interconnections*, pp. 257-263.

Dao, B.V., Yalamanchili, S., and Duato, J. (1997). Architectural Support for Reducing Communication Overhead in Multiprocessor Interconnection Networks. *Proc. 3rd Int'l Symp. on High Perf. Comp. Architecture*, pp. 343- 352.

Dongarra, J.J. and Dunigan, T. (1997). Message-Passing Performance of Various Computers. *Concurrency*, Vol. 9, No. 10, pp. 915-926.

Eicken, T.V., Culler, D.E., Goldstein, S.C., and Schauser, K.E. (1992). Active Messages: A Mechanism for Integrated Communication and Computation. *Proc. of the 19th Ann. Int'l Symp. on Computer Architecture*, pp. 256-265.

Hioki, S. (1996). Construction of Staples in Lattice Gauge Theory on a Parallel Computer. *Parallel Computing*, 22-10, pp. 1335-1344.

Kim, J. and Lilja, D.J. (1998). Characterization of Communication Patterns in Message- Passing Parallel Scientific Application Programs. *Proc. of the Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing. Int'l Symp. on High Performance Computer Architecture*, pp. 202-216.

de Lahaut, D.G. and Germain C. (1994). Static Communications in Parallel Scientific Programs. *Proc. of PARLE '94, Parallel Architecture and Languages.*

Louri, A. and Sung, H.K. (1994). An Optical Multi-Mesh Hypercube: A Scalable Optical Interconnection Network for Massively Parallel Computing. *J. of Lightwave Technology*, Vol. 12, No. 4, pp. 704-716.

Marchand, P.J., Krishnamoorthy, A.V., Yayla, G.I., Esener, S.C., and Efron, U. (1997). Optically Augmented 3-D Computer: System Technology and Architecture. *J. of Parallel and Distributed Computing, Special Issue on Optical Interconnects*, Feb. 25, pp. 20-35.

Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. (1995). Version 1.1.

Mowry, T. and Gupta, A. (1991). Tolerating Latency Through Software-Controlled Prefetching in Shared-Memory Multiprocessors. *J. of Parallel and Distributed Computing*, 12(2), pp. 87-106.

Nordin, R.A., Levi, A.F., Nottenburg, R.N., O'Gorman, J., Tanbun-Ek, T., and Logan, R.A. (1992). A System Perspective on Digital Interconnection Technology. *IEEE J. of Lightwave Technology*, Vol. 10, pp. 801-827.

Pakin, S., Lauria, M., and Chien, A. (1995). High Perf. Messaging on Workstation: Illinois Fast Messages (FM) for Myrinet. *Proc. of Supercomputing '95.*

Panajotov, K., Nieuborg, N., Goulet, A., Veretennicoff, I., and Thienpont, H. (1998). A Free- space Reconfigurable Optical Interconnection based on Polarization-Switching VCSEL's and Polarization-Selective Diffractive Optical Channels. *Proc. of Optics in Computing*, pp. 151-154.

Sakr, M.F., Levitan, S.P., Chiarulli, D.M., Horne, B.G., and Giles, C.L. (1997). Predicting Multiprocessor Memory Access Patterns with Learning Models. *Proc. of the 14th Int'l Conf. on Machine Learning.* pp. 305-312.

Worley, P.H. and Foster, I.T. (1994). Parallel Spectral Transform Shallow Water Model: A Runtime-tunable parallel benchmark code. *Proc. of the Scalable High Performance Computing Conference.* pp. 207-214.

Yayla, G.I., Marchand, P.J., and Esener, S. C. (1998). Speed and Energy Analysis of Digital Interconnections: Comparison of On-chip, Off-chip and Free-Space Technologies. *App. Optics*, Vol. 37, No. 2, pp. 205-227.

Yuan X., Melhem, R., and Gupta R. (1996). Compiled Communication for All-Optical TDM Networks. *Proc. of Supercomputing '96.*

Zhang. Z. and Torrellas, J. (1995). Speeding Up Irregular Applications in Shared-Memory Multiprocessors: Memory Binding and Group Prefetching. *Proc. of the 22nd Ann. Symp. on Computer Architecture*, pp. 188-199.