Hiding Communication Latency in Reconfigurable Message-Passing Environments

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 is one of the most important factors affecting the performance of message passing multicomputers. We present evidence (through the analysis of several parallel benchmarks) that there exists communications locality, and that this locality is "structured". We have devised a number of heuristics that can "predict" the target of subsequent communication requests. This technique, can be applied to reconfigurable interconnects to hide the communications latency by reconfiguring the interconnect concurrently to the computation. By comparing the inter-communication computation times of a number of parallel benchmarks with some specific reconfiguration times, we argue that the computation interval can be used to hide the concurrent reconfiguration of the interconnect, and present the performance enhancements of the proposed heuristics.

1.0 Introduction

Message-passing multicomputers are composed of a number of computing nodes that communicate by exchanging messages through an interconnect. Optics is ideally suited for implementing interconnection networks because of its superior characteristics over electronics [20,15]. Various optical interconnection networks including the works in [5,12] have been proposed.

We have introduced [1] a *reconfigurable optical network*, *RON* (*k, N*), consisting of *N* computing nodes. A node is capable of connecting directly to any other node and can establish *k* simultaneous connections. Connections are established 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.

Various implementation technologies exist to embody the above abstract model. Such technologies include 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 shall assume that one or more of these technologies will be used to implement the proposed interconnect. Under such an implementation, the various overheads associated with the reconfiguration of the network are lumped together as the reconfiguration delay *d*.

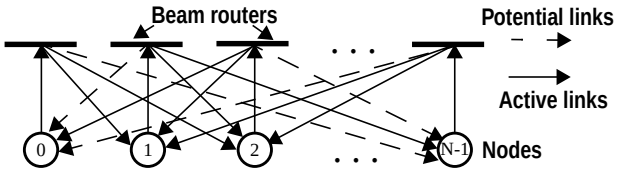

FIGURE 1. ***RON (k, N)*, a massively parallel computer interconnected by a complete free-space optical network**

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 message, τ the per unit transmission time. The setup time t_s [7] and reconfiguration delay d, are the major contribution of the communication delay T, both being of the order of several μs. Several researchers are working to minimize this cost by user-level messaging techniques such as *active messages* [8] or *fast messages* [16]. In this work we are particularly interested in the techniques that hide the reconfiguration delay, *d*.

It is obvious that if a link is already in place, then the configuration phase does not enter the picture with a commensurate savings in the message transmission time. This can be 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, then it is possible to determine the destinations and the instances that these shall be used [1]. However, if the algorithm is not known, the approach mentioned above 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 [14,18,22]. In the context of message passing programming, 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 the *communications locality* properties of parallel applications [10,11].

By communications locality, we 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 a communication channel will remain established until it is explicitly destroyed.

This work has two parts. The first part is an extension of our work in [3] and explores the effect that a number of heuristics has in predicting the target of a communication request. For these studies, we have used the MPI [13] implementation of the NAS parallel benchmarks suite (NPB) (version 2.3, W class) [4], the Parallel Spectral Transform Shallow Water Model (PSTSWM) parallel benchmark (version 6.2) [19], and the pure QCD Monte Carlo Simulation Code with MPI (QCDMPI) parallel benchmark (version 1.4) [9] on an 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. It is worth mentioning that the proposed heuristics can be used in any circuit-switched networks including the *wave switching* [6] and [21].

The second part considers the execution time of the computation phases of these parallel benchmarks on an IBM SP2 using its high performance switch under the user space mode when we had an exclusive access to the system. We show that computation times, are sufficiently large for reconfigurations, proceeding concurrently with computations, to terminate before the computation, and we present the performance enhancements achieved because of the latency hiding power of the heuristics developed.

Section 2 analyzes the proposed heuristics. In section 3, we obtain the inter-send computation times for the benchmarks, and present the performance enhancements of the proposed heuristics. Finally, we conclude with section 4.

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. Our heuristics would execute on each node of the multicomputer, and predict the destination nodes for communications originating at the node on which they reside. 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.

A number of heuristics were proposed and studied in a previous work [2,3]. These include

1. The *Least Recently Used* (LRU) [10], *First-In-First-Out* (FIFO) and *Least Frequently Used* (LFU) heuristics, all of which maintain a set of *k* message destinations [3]. If the next 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 the adopted LRU, FIFO or LFU strategy.
2. The *Single-cycle* heuristic [3], implements a simple cycle discovery algorithm. Starting with a *cycle-head* node the sequence of requests is logged 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 requested node does not coincide with the predicted one, then a new cycle formation stage commences with the cycle-head being the node that caused the miss.
3. The *Single-cycle2* heuristic [3] is identical to the *single-cycle* heuristic with the addition that during cycle formation, the previously requested node is offered as the predicted node. Both cycle heuristics have a better performance than the LRU, FIFO and LFU heuristics under the single-port assumption.

2.1 Better-cycle and Better-cycle2 heuristics

The performance of the *cycle* heuristics is improved if the previously formed cycles are maintained. In the *Better-cycle* heuristic, we keep the last cycle associated with each cycle-head encountered. In case of a miss, if the prediction offered by the stored cycle associated with the node that caused the miss, is incorrect, then a new cycle formation commences. Otherwise, the stored cycle is used to predict the subsequent requests. The state diagram of this heuristic is shown in Figure 2. This heuristic performs better than the Single-cycle and Single-cycle2 heuristics [2].

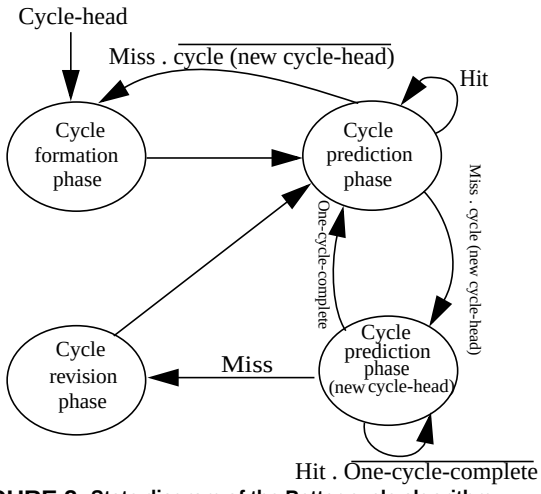

FIGURE 2. State diagram of the Better-cycle algorithm

The *Better-cycle2* heuristic is identical to the *Better-cycle* heuristic with the addition that during the cycle formation and cycle revision phases the previously requested node is offered as the predicted node. The performance of this heuristic is shown in Figure 3. This heuristic has better performance than the *Single-cycle* and *Single-cycle2* heuristics, and the *Better-cycle* heuristic for the BT, SP, and the QCDMPI benchmarks [2].

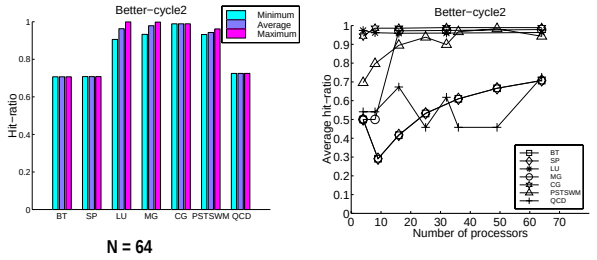

FIGURE 3. Effects of the Better-cycle2 heuristic on the benchmarks

2.2 Tagging heuristic

The *Tagging* heuristic [3], assumes a communications environment in which a particular communication request (send) in a section of code, will be to the same target node with a high 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. This can be implemented with the help of the compiler or by the programmer through a *pre-connect(tag)* operation which will force the communication system to connect to the destination before the actual communications request is issued. This technique is similar to the prefetch operation advocated by Mowry and Gupta [14].

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 communication node is the same as the target already associated with the tag. Otherwise, on a miss, the tag is assigned the newly requested target node.

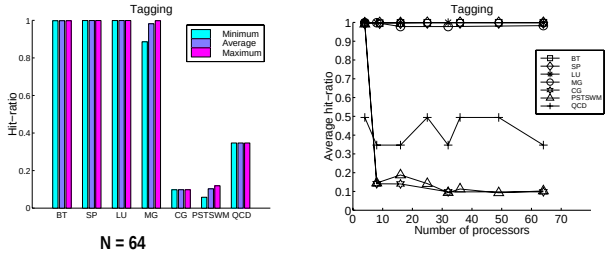

FIGURE 4. Effects of the Tagging heuristic on the benchmarks

The performance of the tagging heuristic is presented in Figure 4. As it can be seen, the tagging heuristic results in an excellent performance (hit ratios in the upper 90%) for all the benchmarks except for 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.

2.3 Tag-bettercycle and Tag-bettercycle2 heuristics

Cycle heuristics are excellent in discovering cyclic occurrences of target. We have combined therefore the *tagging* heuristic with the cycle heuristics discussed in section 2.0. The results for the *tag-cycle* and *tag-cyle2* were presented in [3].

In the Tag-*bettercycle* heuristic, we attach a different tag to each of the communication requests 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, similar to the Better-cycle2 heuristic, the previously requested node is offered as the predicted node. The performance of Tag-bettercycle for the QCDMPI benchmark is better than the Tag-cycle algorithm, but not better than the Tag-cycle2 heuristic [2]. However, the Tag-bettercycle2 heuristic is superior to all other heuristics for all parallel benchmarks, as shown in Figure 5.

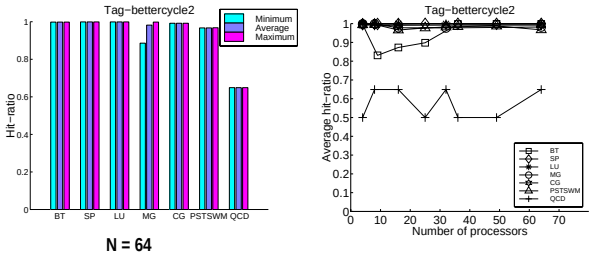

FIGURE 5. Effects of the Tag-bettercycle2 heuristic on the benchmarks

2.4 Heuristics performance comparison

Figure 6, presents a comparison of the performance of the heuristics presented in this work under single-port assumption when the number of processors is 64. It is evident that the Tag-bettercycle2 heuristic is the best for all benchmarks and its hit-ratio is consistently very high for all benchmarks considered here [2].

3.0 Inter-send computation times and Speedup

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

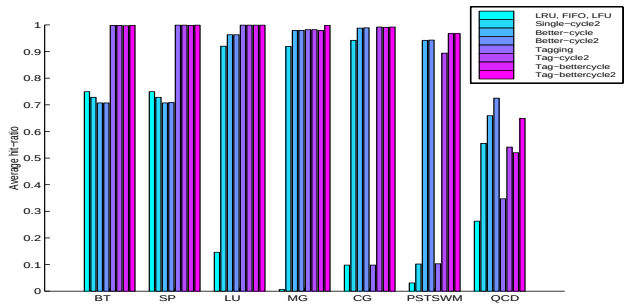

FIGURE 6. Comparison of the performance of the heuristics discussed in this work for the benchmarks under single-port modeling when the number of processors is 64

In Section 2.0 we have presented a number of heuristics that can be used to accurately predict the destination of the subsequent communication request. In this section, we shall argue that, at least in the application benchmarks studied, there is enough computation preceding a communication request to effectively hide the reconfiguration cost.

We used a four node IBM SP2 (160 MHZ P2SC thin nodes and second generation high performance switch) and run the suite of benchmarks in the user space, one process per node, and under an exclusive access to the nodes. 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 pure computation times for each node and for all the benchmarks while Figure 7, presents a summary of the distribution of the inter-send computation times attesting to the fact that the inter-send computation times are distributed widely.

TABLE 1. Minimum inter-send computation times of the parallel benchmarks when N = 4, all times are in microsecond

	Node 0	Node 1	Node 2	Node 3
BT	4.888	4.472	4.472	4.888
SP	4.576	4.264	4.264	4.472
LU	5.824	240.006	600.340	8.840
MG	5.928	6.240	6.240	7.072
CG	335.556	336.908	337.700	365.404
PSTSWM	0.546	0.325	1.027	0.819
QCDMPI	6.448	1166.880	1167.816	1240.34

Researchers in optical engineering are using different approaches to reduce the reconfiguration time of the optical interconnects and are currently reporting reconfiguration times of 25 μs [17]. We compare the pure computation times of these benchmarks with this 25 μs reconfiguration time, and with reconfiguration times of 10, 5, and 1 μs as a measure of future advancements in this area. Figure 7 presents the percentage of the number of computation times less than and more than 1, 5, 10, and 25 μs 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. For the cases where the computation time is not sufficiently long to completely hide the reconfiguration it effectively reduces the reconfiguration cost by the corresponding length of time.

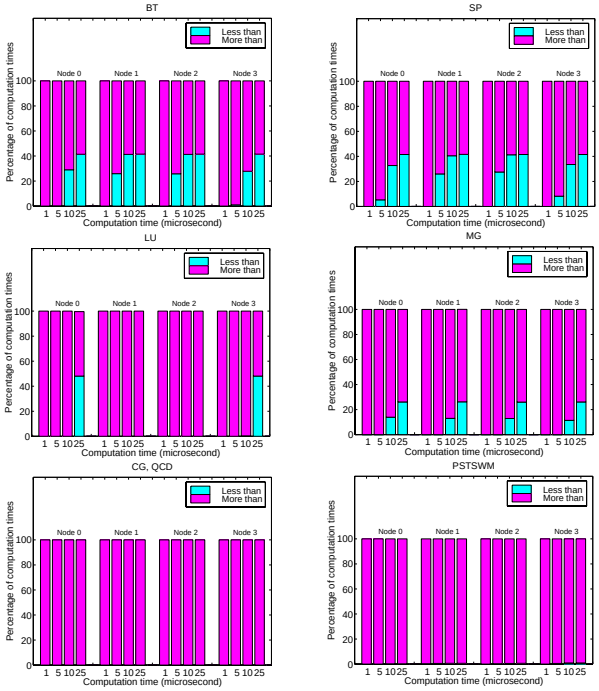

FIGURE 7. Percentage of the inter-send computation times for different benchmarks less than, and more than specific length of times when N = 4

3.1 Total reconfiguration time and application speedup

In this section, we shall quantify the ability of the proposed heuristics to hide the reconfiguration delays. In addition, we shall determine the impact of the proposed heuristics on the application speedup.

We assume a multicomputer with nodes similar to the thin nodes of an IBM SP2 system but with a reconfigurable optical interconnect which has a reconfiguration delay *d* (*d* = 25, 10, 5, 1 μs).

For the calculations that quantify the latency hiding capabilities of our heuristics, we use the lower bound of the inter-send computation times [2]. This allows us to compute the lower bound of the time that can be hidden. For the application speedup computations though, we use the full benchmark trace obtained and the inter-send time includes

overhead associated with the waiting on receive [2]. The algorithm to obtain the time spent in reconfiguring the interconnect with and without the prediction heuristics is given by the following pseudocode.

```
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 8, illustrates the average ratio of the total new reconfiguration delay over the total original reconfiguration delay for each benchmark under two different CPU speeds and four different reconfiguration delays for some of the heuristics [2]. It is clear that the tag-based heuristics have the best performance enhancements. For the CG, LU, and PSTSWM benchmarks, almost all of the reconfiguration delay can be hidden under tag-based heuristics. These results are still preliminary. We are in the process of using a larger system to obtain more realistic traces.

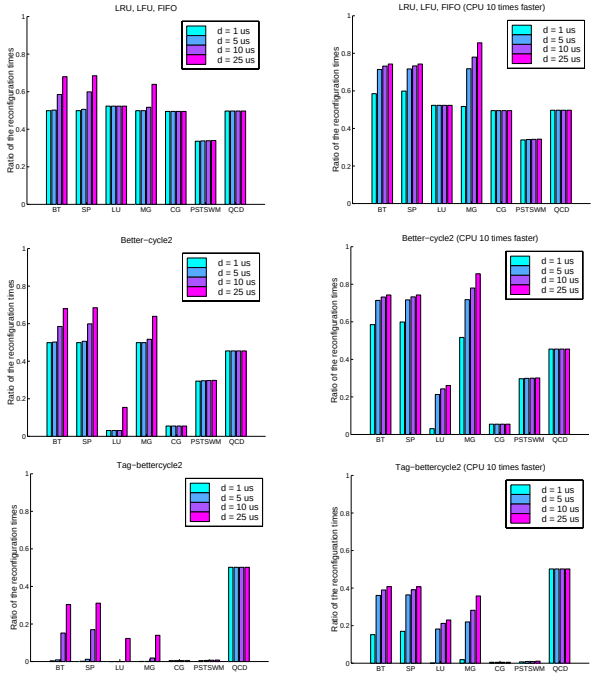

FIGURE 8. The total reconfiguration time obtained while employing the prediction heuristics as a percentage of the total configuration time when no prediction heuristics were employed. Benchmarks assumed a current generation and a 10 times faster CPU when *d* = 1, 5, 10, and 25 microseconds, and *N* = 4 under single-port modeling

In Table 2, we present the maximum application speedup that we can obtain by applying the heuristics. The application speedup has been computed for node 0. The results are consistent with the fact that the total communication time is much smaller than the total computation time. This is because the granularity of the benchmarks on a four node machine is quite large. It is interesting to see what would be the application speedup with a larger machine size and problem size

TABLE 2. Speedup of the benchmarks when n=4 and d=25 μs

Benchmark	Speedup	Benchmark	Speedup
BT	1.0005	CG	1.004
SP	1.0003	PSTSWM	1.0001
LU	1.002	QCDMPI	1.0001
MG	1.006		

4.0 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 publicly available NAS suite, the PSTSWM and the QCDMPI parallel benchmarks. The heuristics proposed are only possible because of the existence of *communications locality* and can be used to establish a communications 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.

In the second part of our work, we presented the execution times of the computation phases of these parallel benchmarks on an IBM SP2 using its high performance switch under the user space when we had exclusive access to the system. 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 inter-send computation phases. 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/communication traces and heuristics hit/miss profiles to determine the total reconfiguration time under different reconfiguration costs

and processor speeds. The results indicate that the tag-better cycle2 heuristic has the best performance.

The speedup results are explainable since the execution traces were obtained on a four node machine. A four node system is heavily compute bound. Any improvements on the communication have minimal effect due to Amdahl's law. We are planning to extend our measurements to systems with more than four nodes where the computation and communication are balanced.

We expect 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 reconfiguarability and fast optical fabrics.

Acknowledgments

This work was supported by grants from NSERC and the University of Victoria. Special thanks to the staff of the computer center at the University of Victoria for their kind cooperation in accessing the university IBM SP2. We also want to thank the staff of IBM Victoria for their help. Finally, we would like to thank the anonymous referees for their comments.

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, "Hiding Communication Latency in Reconfigurable Message-Passing Environments," Technical Report ECE-99-3, Department of Electrical and Computer Engineering, University of Victoria, Jan., 1999

[3] A. Afsahi and N. J. Dimopoulos, "Communications Latency Hiding Techniques for a Reconfigurable Optical Interconnect: Benchmark Studies," Proceedings of PARA98, Fourth International Workshop on Applied Parallel Computing, Large Scale Scientific and Industrial Problems, June, 1998

[4] D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, "NAS Parallel Benchmark Result 3-94," Proceedings of the Scalable High-Performance Computing Conference, 1994, pp. 111- 120

[5] H. Bourdin, A. Ferriera, and K. Marcus, "A Comparative Study of One-to-Many WDM Lightwave Interconnection Networks for Multiprocessors," Proceedings of the Second International Conference on Massively Parallel Processing using Optical Interconnections, 1995, pp. 257-263

[6] B. V. Dao, Sudhakar Yalamanchili, and Jose Duato, " Architectural Support for Reducing Communication Overhead in Multiprocessor Interconnection Networks," Proceedings of the Third International Symposium on High Performance Computer Architecture, 1997, pp. 343-352

[7] J. J. Dongarra and T. Dunigan, "Message-Passing Performance of Various Computers," Concurrency, Vol. 9, No. 10, Dec. 1997, pp. 915-926

[8] T. V. Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," Proceedings of the 19th Annual International Symposium on Computer Architecture, May 1992, pp. 256-265

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

[10] J. Kim and D. J. Lilja, "Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs," Proceedings of the Workshop on Communication, Architecture, and Applications for Network-based Parallel Computing, International Symposium on High Performance Computer Architecture, Feb. 1998, pp. 202-216

[11] D. G. de Lahaut and C. Germain, "Static Communications in Parallel Scientific Programs" Proceedings of PARLE'94, Parallel Architecture and Languages, July 1994, pp. 262-276

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

[13] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. Version 1.1 (June 1995)

[14] 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

[15] R . A. Nordin, A. F. Levi, R. N. Nottenburg, J. O'Gorman, T. Tanbun-Ek, and R. A. Logan, "A System Perspective on Digital Interconnection Technology," IEEE Journal of Lightwave Technology, Vol. 10, June 1992, pp. 801-827

[16] S. Pakin, M. Lauria, and A. Chien, " High Performance Messaging on Workstation: Illinois Fast Messages (FM) for Myrinet," Proceedings of Supercomputing'95, Nov., 1995

[17] K. Panajotov, N. Nieuborg, A. Goulet, I. Veretennicoff and H. Thienpont, "A Free-space Reconfigurable Optical Interconnection based on Polarization-Switching VCSEL's and Polarization-Selective Diffractive Optical Channels," Proceedings of the Optics in Computing, 1998, pp. 151-154

[18] M. F. Sakr, S. P. Levitan, D. M. Chiarulli, B. G. Horne, and C. L. Giles, "Predicting Multiprocessor Memory Access Patterns with Learning Models," Proceedings of the Fourteenth International Conference on Machine Learning," 1997, pp. 305-312

[19] P. H. Worley and I. T. Foster, "Parallel Spectral Transform Shallow Water Model: A Runtime-tunable parallel benchmark code," Proceedings of the Scalable High Performance Computing Conference, 1994, pp. 207-214

[20] 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. 1998, pp. 205-227

[21] X. Yuan, R. Melhem and R. Gupta, "Compiled Communication for All-Optical TDM Networks," Proceedings of the Supercomputing'96, 1996

[22] Z. Zhang and J. Torrellas, "Speeding Up Irregular Applications in Shared-Memory Multiprocessors: Memory Binding and Group Prefetching," Proceedings of the 22nd Annual Symposium on Computer Architecture, 1995, pp. 188-199