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

Ahmad Afsahi Nikitas J. Dimopoulos

Technical Report ECE-98-2

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

June, 1998

Abstract. *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. The communication overhead is one of the most important factors affecting their performance. In this work, we present evidence (through the analysis of several parallel benchmarks) that there exists communication locality, and that this locality is "structured". We have utilized this structure in a number of heuristics that can be used to "predict" the target of subsequent communication requests. This technique, can be applied directly to reconfigurable optical interconnects to hide the communications latency by reconfiguring the communications network concurrently to the computation. This method has applicability to conventional interconnection networks as well.*

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

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. As the communication overhead is one of the most important factors affecting their performance, there has been a great deal of interest in the design of the interconnection networks. In this respect, various types of static and dynamic interconnection networks, such as complete networks, hypercubes, meshes, rings, tori, omega, and butterfly have been proposed and some of them have been implemented.

A complete interconnection network, where any computing node can communicate with any other node in a single hop is very desirable. Complete interconnection networks can be modeled by a complete graph K_N, where all N vertices are linked together and the diameter is one. Each vertex has degree N - 1 and the number of edges is $N(N-1)/2$, far too high to be practical when N is large. These limitations prevent implementing complete networks using metal-based interconnections. However, the use of optical technology to interconnect a large number of electronic processors is very promising for the construction of high-performance computers in the future.

Optics is ideally suited for implementing interconnection networks because of its superior characteristics over electronics [19][16][9], such as higher interconnection density, higher bandwidth, suitability for reconfigurable interconnects, larger number of fan-in and fan-out, lower error rate, freedom from planar constraints (light beams can easily cross each other), immunity from electromagnetic field and ground loops, lower signal crosstalk etc. It is foreseen that future massively parallel machines will employ optical interconnections in an ever increasing manner.

In free-space optical interconnects, optical signals can propagate very close to each other and pass each other without interaction. One of its main features is the capability to reconfigure the interconnections on the demand which is very suitable for the construction of 3-D VLSI computers [13]. Free-space optical interconnects use free-space (vacuum, air or glass) for optical signal propagation and include two basic components: one is the optoelectronic device for photon generation or modulation such as VCSELs and SEEDs, while the other component is the optical beam router to redirect or distribute optical beams. Optical beam routing in a free-space optical interconnection network often employs external optical elements such as holograms, mirrors, prisms, lenses etc.

Research in massively parallel computers using optical interconnections is focusing in two major directions; One is to implement traditional, and design new novel, point-to-point interconnection networks [12] but utilizing the advantages of the new emerging optical technologies to overcome the constraints of metal-based communication, while the second direction is to design new interconnection networks [3] based on the constraints of optical devices, such as optical passive star coupler (OPS) interconnection networks which can be modeled by hypergraphs.

Along the first direction, we have introduced [1] a complete network using a reconfigurable free-space optical interconnect. We define an abstract model [1] for our complete free-space optical interconnection network for massively parallel computers, and discuss its characteristics.

Definition A *reconfigurable optical interconnection network*, OK_N, consists of N computing nodes with their own local memory. A node is capable of connecting directly to any other node. 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 1. Messages are sent using circuit-switching. That is, a connection must be established between the source and destination pair before the message is sent. Each node has the ability to simultaneously send and receive k messages on its k links (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.

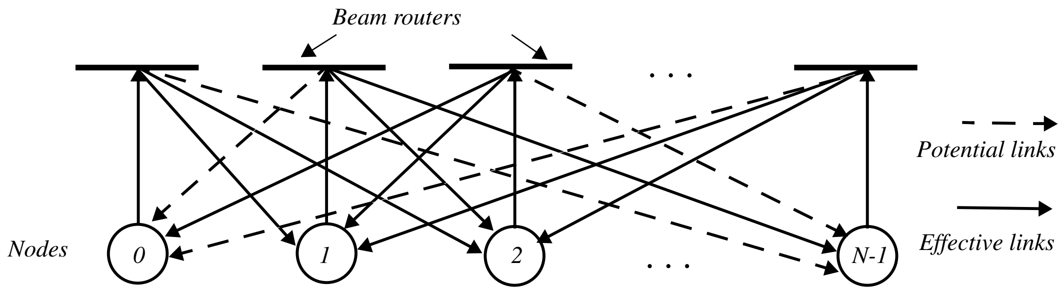


FIGURE 1. OK_N, a massively parallel computer interconnected by a complete free-space optical interconnection network

Various implementation technologies exist to embody the above abstract model. Such technologies include computer generated holograms and/or deformable mirrors for switching, frequency hoping for coding, wavelength tuning for transceivers, VCSELs and/or SEEDs for photon generation or modulation. In this paper, we are particularly interested in the abstract model of our proposed network and do not address these implementation issues. We shall assume that one or more of the technologies outlined above will be used to implement the proposed interconnect. Under such an implementation, the various overheads associated with the reconfiguration of the network (such as beam steering, setting up the computer-generated holograms, tuning the transmitters, or sending the frequency code in a frequency hoping implementation etc.) are lumped together as the (re)configuration delay d.

An important concern is to model 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 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 (d is an integer) 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 in the order of 10 microseconds [6]. Several researchers are working to minimize this cost by using active messages [7] and fast messages [17]. In this work, we are particularly interested in the techniques that hides the reconfiguration delay, d, so that we can share the costs of message transmission across multiple message transmissions.

We shall assume that a node sends a message to another node by first establishing a link to the target (hence the configuration 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. In this way, the communication pathway could be established and be ready to be used as soon as the message to be sent becomes available.

There are several ways of accomplishing this. If the communication operation is regular and known, then it is possible that one can determine the destinations and the instances that these shall be used. We have developed such algorithms for broadcasting and have reported these in [1]. However, if the algorithm is not known, regular or simple, the approach mentioned above cannot be used.

Many parallel algorithms are built from loops consisting of computation and communication phases. Therefore, communication patterns may be repetitive. This has motivated researchers to find the *communications locality* properties of parallel aplications [10][11]. T. Mowry and A. Gupta [15] have used prefetching, multithreading, and caching to hide/reduce the latency in shared memory multiprocessors. Authors in [18] have used neural networks to predict the next memory request in shared memory multiprocessor.

We need to define the terms *communication locality* and *caching* in conjunction to this work. 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. J. Kim and D. J. Lilja [10] have recently shown that there is a locality in message destination, message sizes, and consecutive runs of send/receive primitives in parallel algorithms. They have proposed and expanded the concept of memory access locality based on the *Least Recently Used*, LRU, stack model to determine these localities.

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. In addition, 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. As we already mentioned, in the context of free-space optical interconnect maintaining a communications channel established does not interfere with communications that are in progress in other parts of the network.

The main focus of our work is to explore the effect that a number of heuristics has in predicting the target of a communications 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 network interface or off-line in the compilation phase. It is worth mentioning that these heuristics can be used in any circuit-switched network including the works proposed in [5][20] in which the authors exploit the communications locality to improve the performance of the parallel computers in the circuit-switched electronics and optical interconnection networks, respectively. In the following section, we shall discuss the benchmarks used. Section 3 outlines the target prediction heuristics used and presents their performance in terms of the parallel benchmarks, and we finally conclude with section 4.

2.0 Parallel Benchmarks.

In order to study the performance of our algorithms, we use some well-known parallel benchmarks form the NAS parallel benchmarks suite (NPB) [2]. These benchmarks have been developed at NASA Research Center to study the performance of parallel supercomputers and networks of workstations. The NAS parallel Benchmarks (NPB) are a set of eight benchmark problems, each of which focuses on some important aspect of highly parallel supercomputing for aerophysics applications. The NPB consists of five “kernels”, and three “simulated computational fluid dynamic (CFD) applications”. In this paper, we are interested only in the communication patterns of point-to-point communication between pair-wise nodes. 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.

The conjugate gradient kernel, CG, tests the performance of the system for unstructured grid computations which by their nature require irregular communications which is a challenge for all kinds of parallel computers. In this kernel, the inverse power method is used to find an estimate of the largest eigenvalue of a symmetric positive-definite sparse matrix with a random pattern of non-zeros. The second kernel benchmark is a simplified multigrid kernel, MG, which solves a 3-D poisson PDE. This code is a good test of both short and long distance highly structured communication. The three simulated CFD application benchmarks, LU, SP, and BT are intended to accurately represent the principal computational and data movement requirements of modern CFD applications. The lower-upper diagonal benchmark, LU, employs a symmetric successive over-relaxation (SSOR) numerical scheme to solve a regular-sparse block 5×5 lower and upper triangular system. In the scalar pentadiagonal benchmark, SP, multiple independent systems of non-diagonally dominant, scalar pentadiagonal equations with a 5×5 block size are solved. In the block tridiagonal benchmark, BT, multiple independent systems of non-diagonally dominant, block tridiagonal equations with a 5×5 block size are solved.

We have used the MPI [14] implementation of NAS benchmarks (version 2.3, workstation class) on a network of SUN4, SUN Solaris, and HPUX workstations using MPICH [8] and Ethernet as the communication medium. We wrote our own profiling code using the wrapper generator facility of MPI available in MPICH to gather the communication traces of these applications when the number of processes are 4, 8, 16, and 32 for LU, MG, and CG benchmarks, and 4, 9, 16, 25, and 36 for BT, and SP. It is worth mentioning that gathering traces does not affect the communication patterns of these applications.

In what follows, we first review the performance of the LRU algorithm [10] on the benchmarks and then present new algorithms employing different heuristics to hide the communications latency on our reconfigurable optical interconnect. Although the derived results are for the above mentioned parallel benchmarks we believe that these benchmarks are representative of the existing parallel applications.

3.0 Latency hiding heuristics

The set of heuristics proposed in this section predict the destination node of a subsequent communication request (MPI_send and MPI_Isend in our applications) based on a past history of communication patterns on a per source node basis. 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. Although our heuristics are applicable to any-port model, we shall present most of our results under the single port communications model where a source node can connect to a single destination node at a time.

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. In the following figures, the number of processors is 36 for BT and SP, and is 32 for LU, MG, and CG, unless otherwise stated.

3.1 The LRU heuristic

The Least Recently Used (LRU) stack model of message destination locality assumes a stack to contain the *k* most recent message destination nodes for each source node where *k* is called the window size of this stack. The window size corresponds to the number of ports used in the communication modeling. Thus, a window size of 1 corresponds to a single-port model while a window size of *k* corresponds to a *k*-port model. For each communication request, if the destination node is already in the stack we have a *hit*, otherwise a *miss* is recorded. Figure 2, shows the result of the LRU heuristic on our benchmarks. Figure 3, shows the scalibiltiy of the LRU heuristic on the benchmarks.

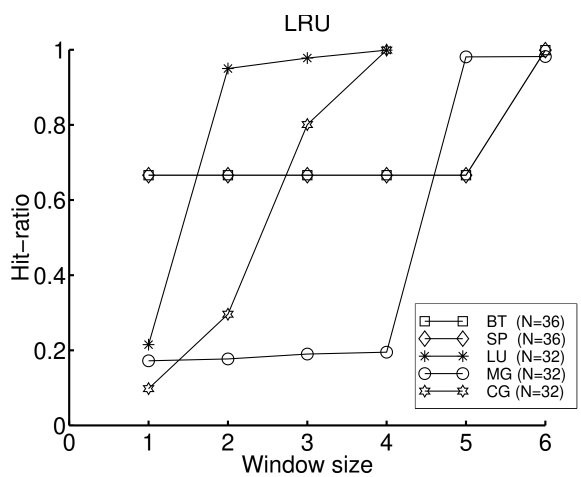


FIGURE 2. Effects of the LRU heuristic on the NAS benchmarks

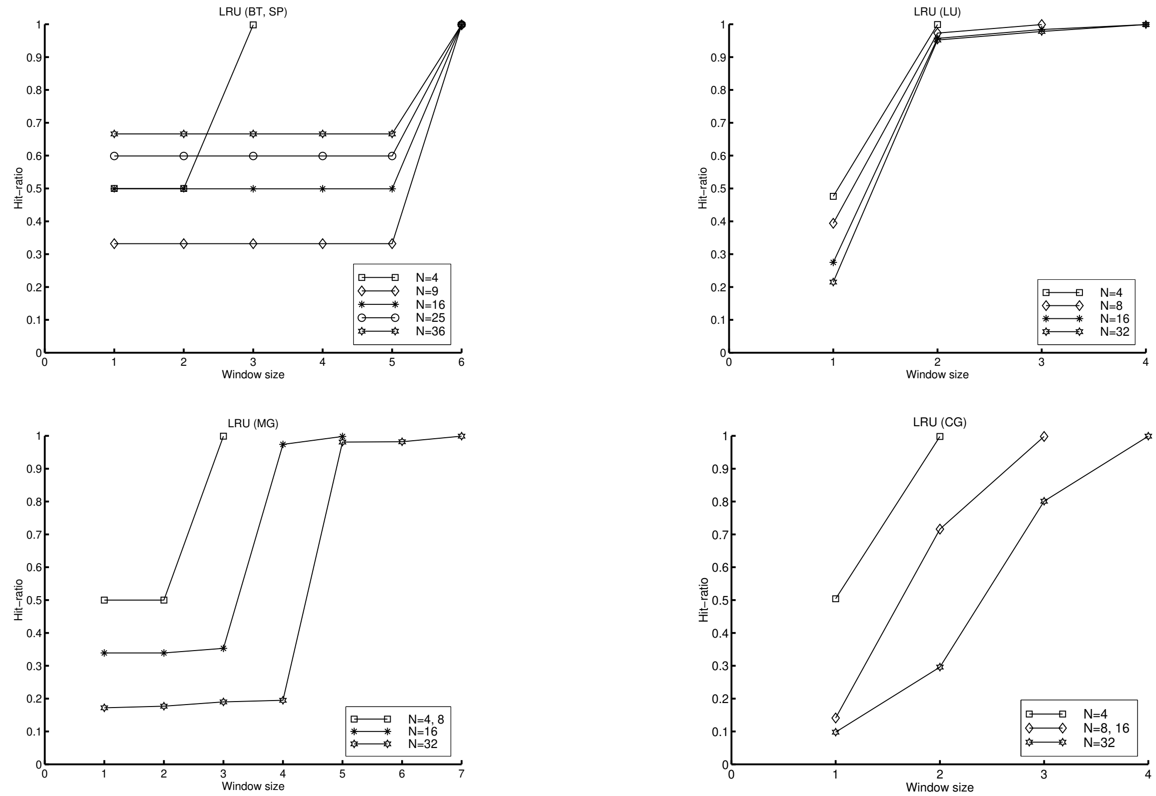


FIGURE 3. Scalibility of the LRU heuristic on the the NAS benchmarks

3.2 The FIFO heuristic

This algorithm associates a time with each message destination when it is brought into the memory. This can be easily implemented by a *First-In-First-Out* (FIFO) queue to hold all the nodes for a window size of *k*. New destinations are inserted at the head of the queue. Old destinations are eliminated from the tail of the queue. If the destination is already in the queue, it moves to the head of the queue. The performance of this algorithm is almost the same with LRU for all benchmarks. Figure 4, shows the result of the FIFO heuristic on our benchmarks. Figure 5, shows the scalibiltiy of the FIFO heuristic on the benchmarks.

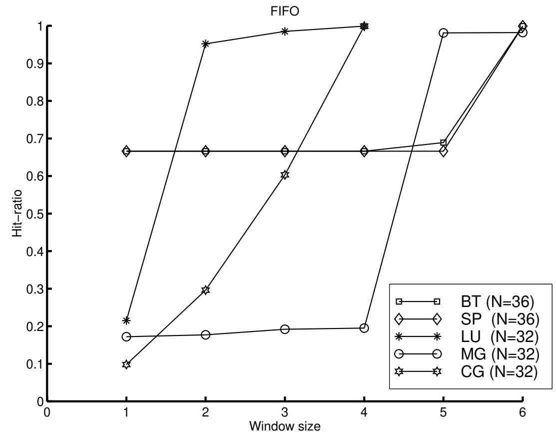


FIGURE 4. Effects of the FIFO heuristic on the NAS benchmarks

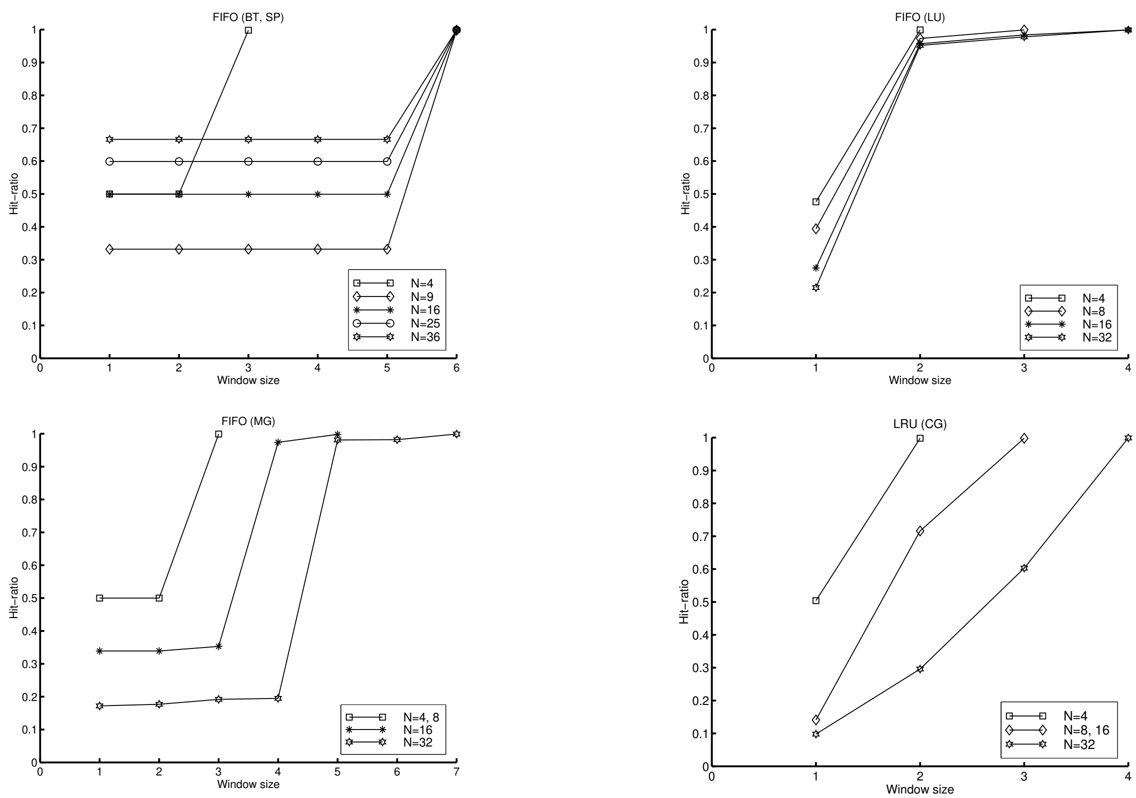


FIGURE 5. Scalibility of the FIFO heuristic on the NAS benchmarks

3.3 The LFU heuristic

The *Least Frequently Used* (LFU) algorithm also maintains a set of k destination nodes. With each destination node, one associates a *use-counter*. Every time a destination node in the set is used, its *use-counter* is incremented by one. If a destination node which is not in the set is requested, then the destination node with the smallest *use-counter* is removed from the set and it is replaced by the requested one. A hit is recorded if the requested destination node is in the set, otherwise a miss is recorded. The results show that the LFU algorithm is better than LRU and FIFO, the exception is the LU benchmarks for $k = 2$. Figure 6, shows the result of the LFU heuristic on our benchmarks. Figure 7, shows the scalibiltiy of the LFU heuristic on the benchmarks.

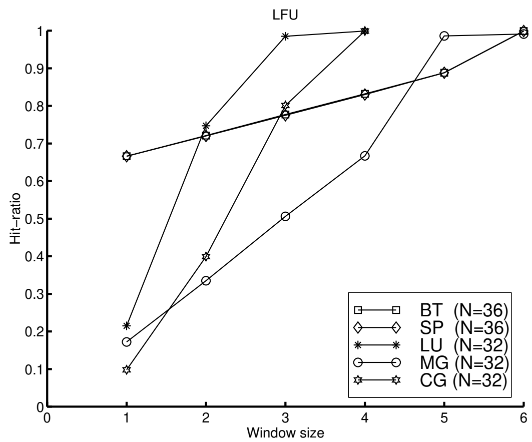


FIGURE 6. Effects of theLFU heuristic on the NAS benchmarks

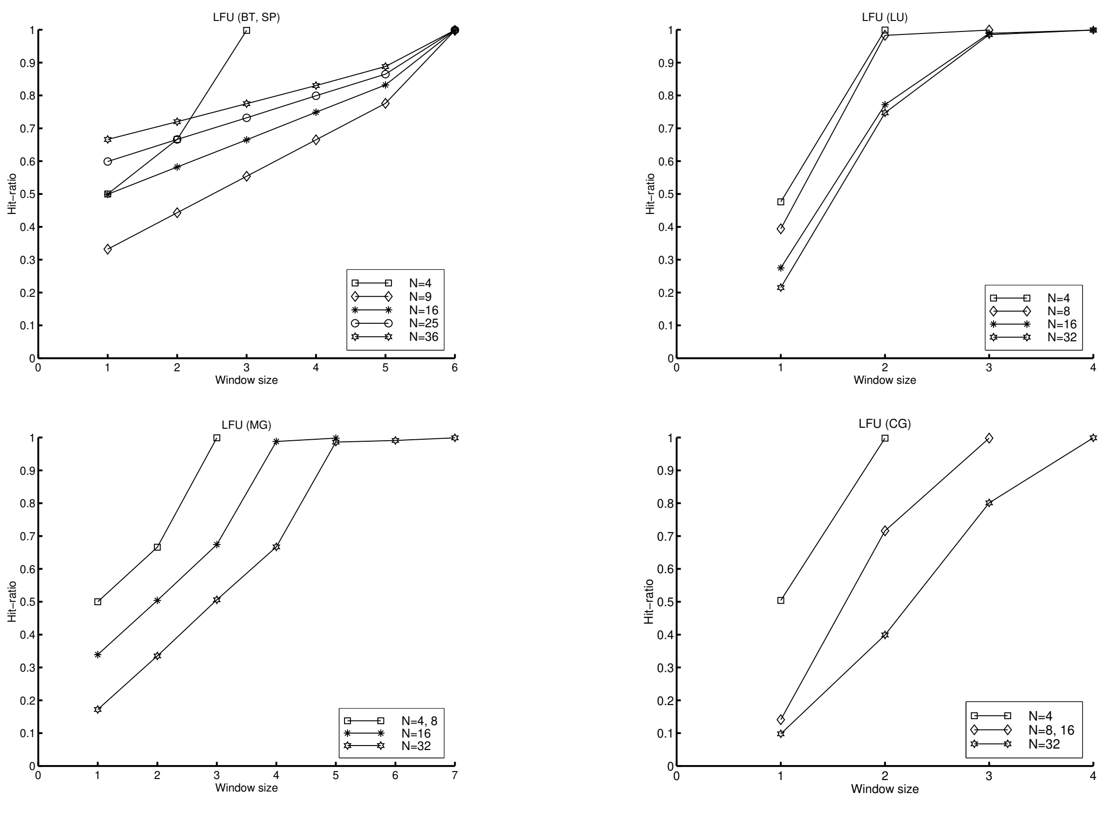


FIGURE 7. Scalibility of the LFU heuristic on the NAS benchmarks

3.4 The Single-cycle heuristic

This heuristic is 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 established as soon as the current request terminates.

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 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.

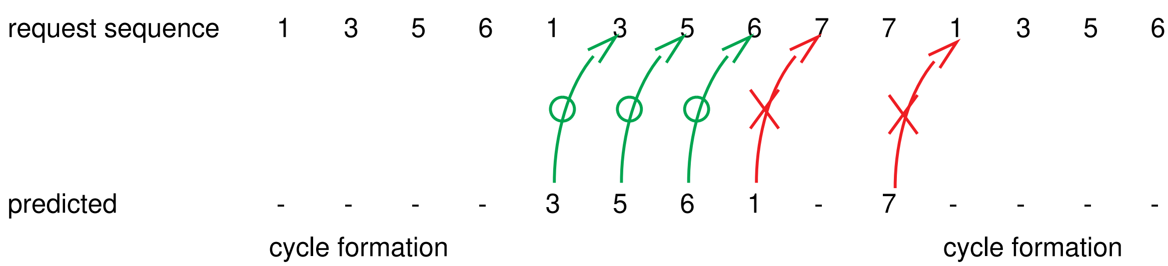


Figure 8, shows the behavior of this algorithm. This algorithm behaves much better than the LRU, FIFO and LFU heuristics for the LU, MG, and CG benchmarks, as shown in figures 2, 4, and 6. However its performance deteriorates for the BT and SP benchmarks. One of the reasons is that in these benchmarks there exist cycles of length one (such as the one composed of node 7 in the figure above). For these situations, there will always be two misses until the heuristic determines that there is a cycle of length 1. The Single-cycle2 heuristic presented next improves the performance for the BT and SP benchmarks without changing the performance for the LU, MG and GC benchmarks.

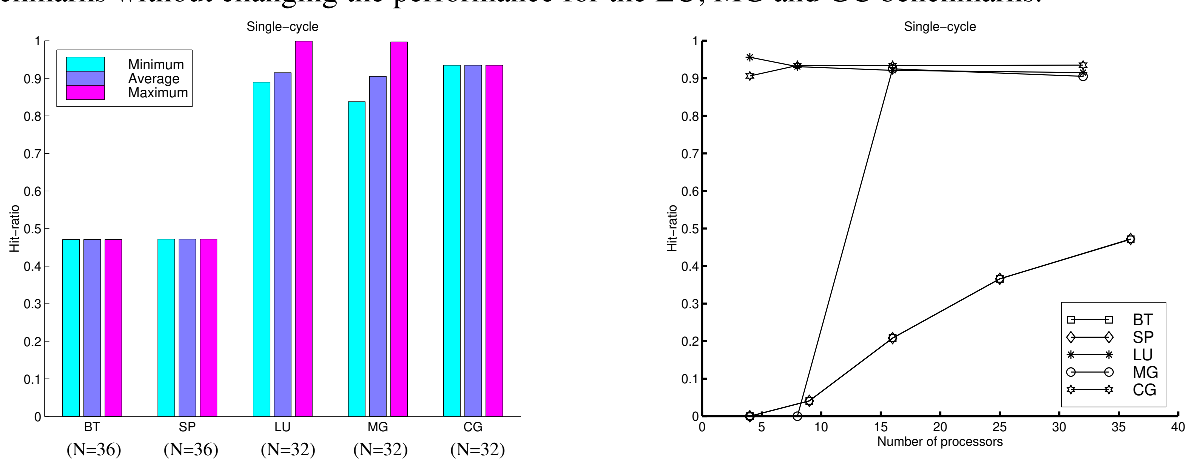
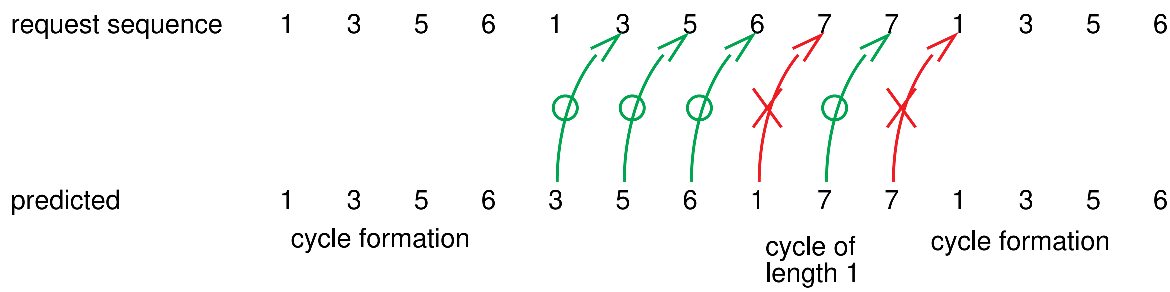


FIGURE 8. Effects of the Single-cycle heuristic on the NAS benchmarks

3.5 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. If a miss occurs during cycle formation, the formation phase continues until a cycle is formed. Then and only then misses cause a new cycle formation phase to begin. We applied the Single-cycle2 heuristic to the request sequence of the previous example. As it was expected, the Single-cycle2 heuristic reacts better to cycles of length 1.



This algorithm has better performance than the single-cycle algorithm especially for the BT and SP benchmarks, as shown in Figure 9. This heuristic performs exceptionally well compared to the LRU, FIFO, and LFU algorithms for the LU, MG, and CG under single-port modelling. However, its hit-ratios for the BT, and SP benchmarks is slightly less than the LRU, FIFO, and LFU algorithms (0.64 compared to 0.66).

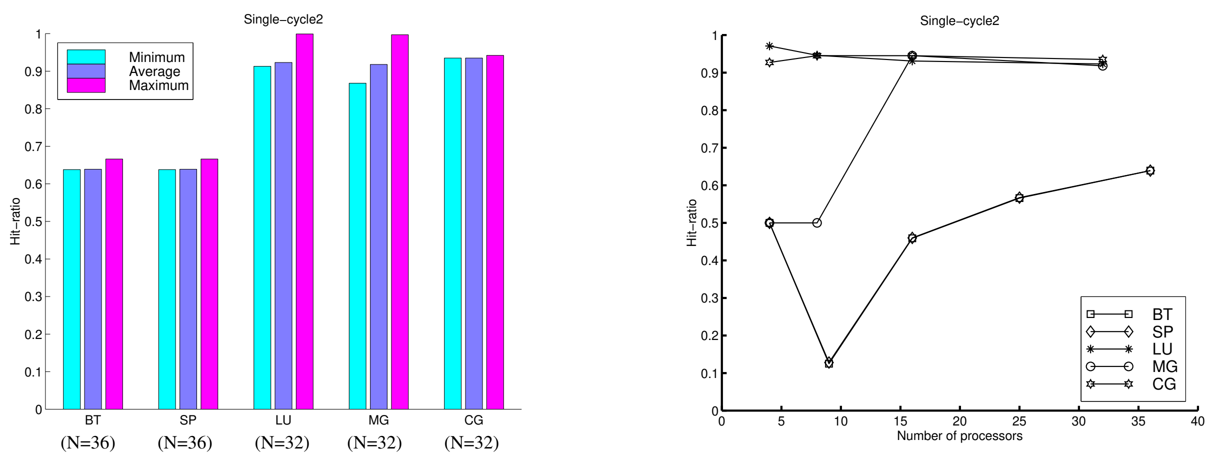


FIGURE 9. Effects of the Single-cycle2 heuristic on the NAS benchmarks

3.6 The History heuristic

The History heuristic maintains a stack of size *k* for each destination node that has been visited. Each stack contains the *k* most recent immediate successors of the node the stack belongs to. The stack is updated using the LRU heuristic. That is, if a node needs to be replaced in the stack, the least recently used node is chosen. A hit is recorded, if the successor of a node is currently in the stack maintained for this node. As an example, assume the following sequence of requests:

Request sequence 1 3 5 1 1 3 5 1 7 7 1 3 5 6

Then the stack with window size of 3 associated with the destination node 1 will contain nodes 3, 1, and 7 at the end of the request sequence as they are the immediate successors of node 1. Figure 10, shows the result of Hisory heuristic on our benchmarks. Figure 11, shows the scalibiltiy of the History heuristic on the benchmarks. The History heuristic performs better than the LRU, FIFO, and LFU heuristics on all benchmarks, except for the BT, and SP benchmarks under single-port modelling where they have the same hit-ratios. Meanwhile, the History heuristic has smaller window sizes. The History heuristic is better than the Single-cycle and Single-cycle2 heuristics for BT, SP, and LU benchmarks. However, the Single-cycle and Single-cycle2 heuristics perform better for the MG and CG benchmarks.

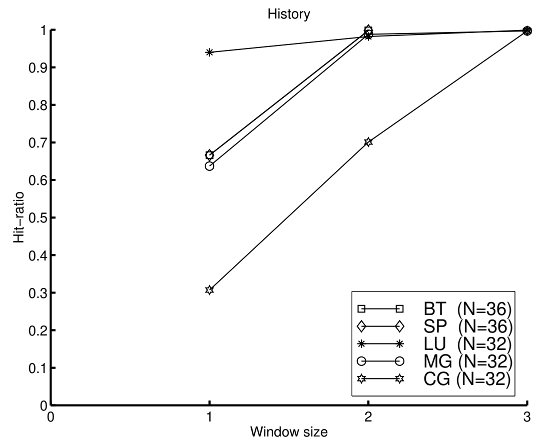


FIGURE 10. Effects of the History heuristic on the NAS benchmarks

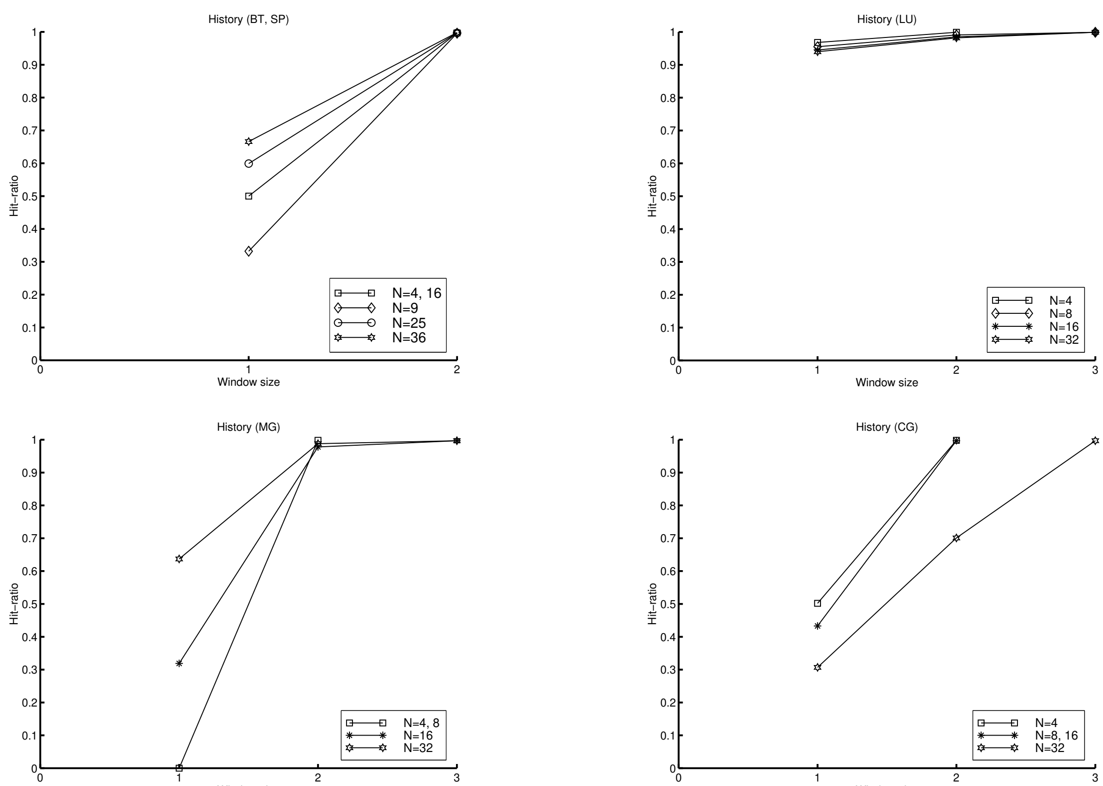


FIGURE 11. Scalibility of the History heuristic on the NAS benchmarks

3.7 The Tagging heuristic

This 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 MC6860 [4].

The performance of the tagging heuristic is presented in Figure 12. As it can be seen, 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 in Section 3.4 and in Section 3.5 the cycle heuristics are excellent in discovering such cyclic occurrences.

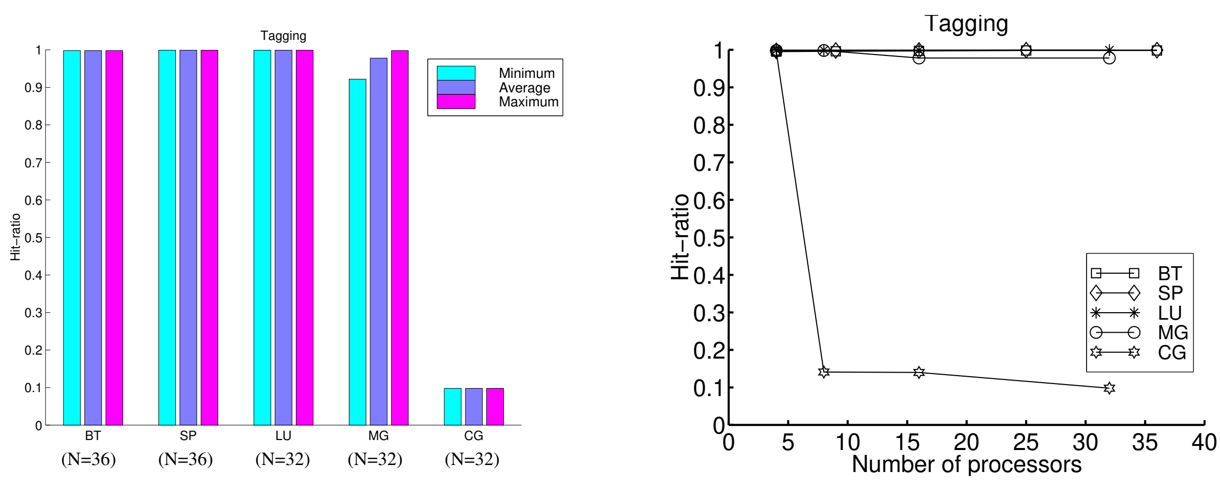


FIGURE 12. Effects of the Tagging heuristic on the NAS benchmarks

3.8 Tag-cycle heuristic

The tagging heuristic does not have a good performance on the CG benchmark while the cycle hueristic showed very good results. Therefore, in order to find a heuristic that performs well across all our benchmarks, we combine the Tagging algorithm with the Single-cycle algorithm and call it the Tag-cycle algorithm.

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[15].

In 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 this tag, we assign the requested target node, to be called *tagcycle-head* node (this is the first destination node that is requested at this tag, or the node that causes a miss). We log the sequence of the requests at this tag until the tagcycle-head node is requested again. This stored sequence constitutes a cycle at each tag, and can be used to predict the subsequent requests. A hit is recorded if in subsequent encounter of the tag, the requested communication node is the same as the predicted node in the cycle. If not, then we record a miss and the cycle formation stage begins with the tagcycle-head being the node that caused the miss. The Tag-cycle heuristic perform exceptionally well across all the benchmarks, as shown in Figure 13.

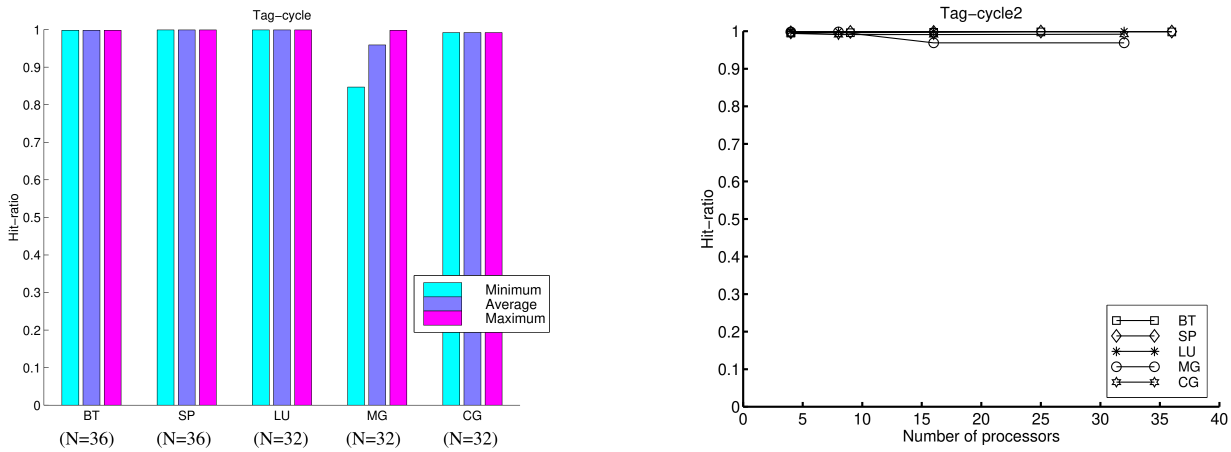


FIGURE 13. Effects of the Tag-cycle heuristic on the NAS benchmarks

3.9 Tag-cycle2 heuristic

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 and is shown in Figure 14.

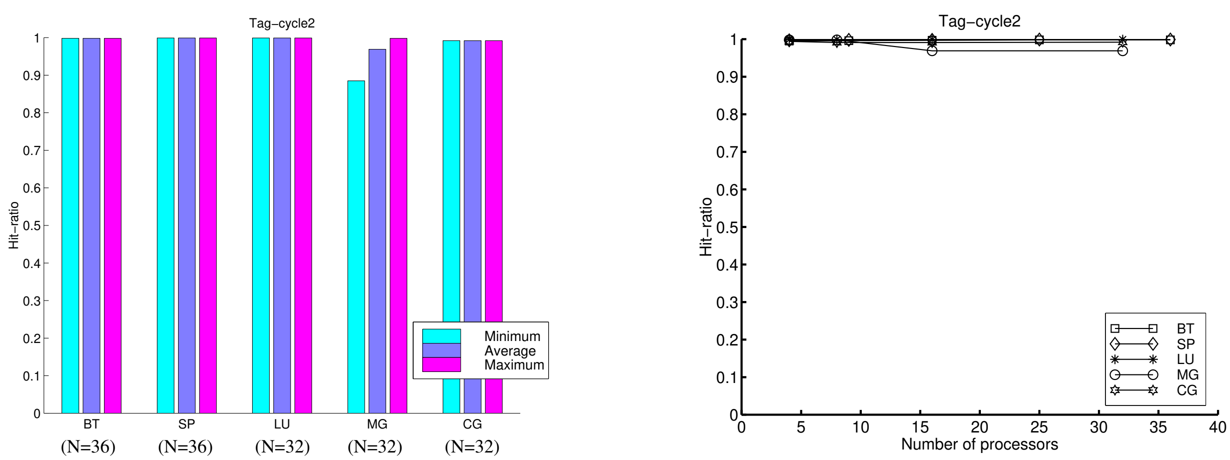


FIGURE 14. Effects of the Tag-cycle2 heuristic on the NAS benchmarks

4.0 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. Our experiments also showed that the nodes in message passing parallel programs (at least in these benchmarks) tend to communicate with only a subset of the other nodes.

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).

Figure 15, 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.

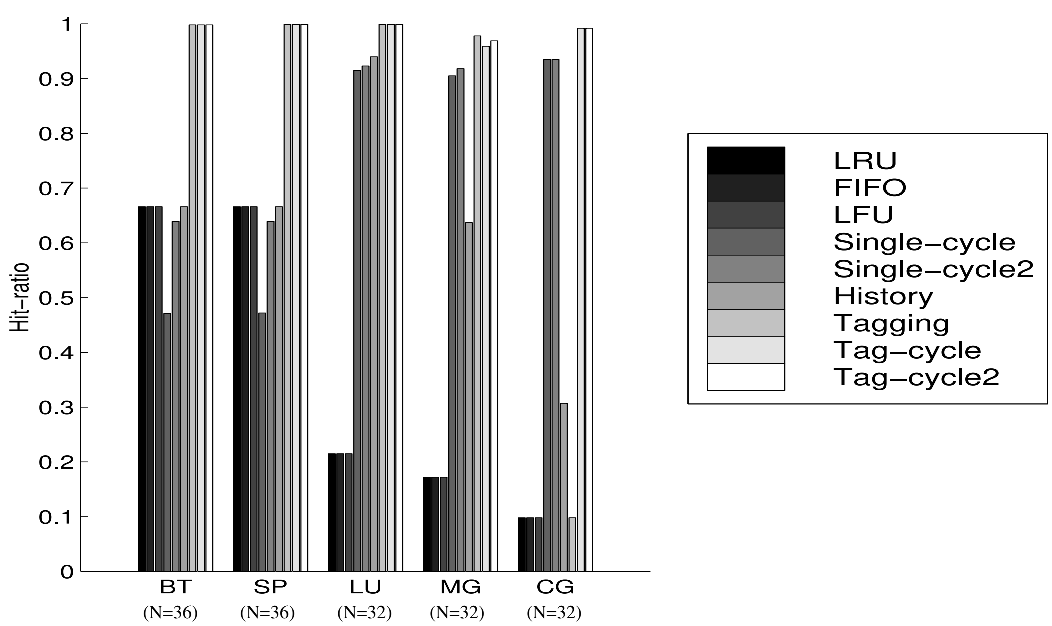


FIGURE 15. Comparison of the performance of the heuristics discussed in this work for the five NAS benchmarks under single-port modelling

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. D. H. Bailey, E. Barszcz, L. Dagum, and H. D. Simon, “NAS Parallel Benchmark Result 3-94,” Proceedings of Scalable High-Performance Computing Conference, 1994, pp. 111- 120
3. 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
4. J. Circello, G. Edgington, D. McCarthy, S. Sullivan, D. Schimke, J. Gay, R. Duerden, C. Hinds, D. Marquette, L, Sood, A. Crouch, D. Chow, “The Superscalar Architecture of the MC68060,” IEEE Micro, Volume 15, Number 2 April 1995, pp. 10-21
5. 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
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, 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
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. M. W. Haney and M. P. Christensen, “Fundamental Geometric Advantages of Free-Space Optical Interconnect,” Proceedings of the Third International Conference on Massively Parallel Processing using Optical Interconnections, 1996, pp. 16-23
10. J. Kim and D. J. Lilja, “Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs,” High-Performance Parallel Computing Research Group, University of Minnesota, Technical Report No. HPPC-97-10, 1997, also in Proceedings of the fourth International Symposium on High Performance Computer Architecture, 1998
11. 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
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. P. J. Marchand, A. V. Krishnamoorthy, G. I. Yayla, S. C. Esener, and U. Efron, “Optically Augmented 3-D Computer: System Technology and Architecture,” Journal of Parallel and Distributed Computing, Special Issue on Optical Interconnects, Feb. 25, 1997, pp. 20-35
14. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard. Version 1.1 (June 1995)
15. 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
16. 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
17. S. Pakin, M. Lauria, and A. Chien, “ High Performance Messaging on Workstation: Illinois Fast Messages (FM) for Myrinet,” Proceedings of Supercomputing’95, Nov., 1995
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. G. I. Yayla, P. J. Marchand, and S. C. Esener,” Energy Requirement and Speed Analysis of Electrical and Free-Space Optical Digital Interconnections,” in Optical Interconections and Parallel Processing: Trends at the Interface. eds.: P. Berthome and Afonso Ferreira, Kluwer Academic Publishers
20. X. Yuan, R. Melhem and R. Gupta, “Compiled Communication for All-Optical TDM Networks,” Proceeding‘s of Supercomputing’96, 1996