Software Techniques for Improving MPP Bulk-Transfer Performance

Eric A. Brewer, Paul Gauthier, Armando Fox, Angela Schuett
University of California at Berkeley

March 18, 2008

Abstract

Brewer and Kuszmaul [BK94] demonstrated how barriers and traffic interleaving can alleviate the problem of bulk-transfer performance degradation on the Thinking Machines CM-5, by exploiting the observation that 1-on-1 communication avoids network congestion. We apply and extend these techniques on the Intel Paragon and MIT Alewife machines. Because these machines lack the CM-5's fast hardware support for barriers, we introduce a token-passing scheme that avoids barriers while maintaining 1-on-1 communication. We also introduce a new algorithm, *distributed dynamic scheduling*, that brings Brewer and Kuszmaul's observations to bear on irregular traffic patterns by massaging traffic into a sequence of near-permutations at runtime, without requiring any preprocessing or global state. The measured performance of our algorithm exceeds that of traffic interleaving (the most effective technique proposed in [BK94]) on all three platforms, and is comparable to the performance of static scheduling, which requires preprocessing and global state [RSA].

1 Introduction

Parallel scientific applications running on massively parallel processors (MPPs) commonly exhibit one of a small set of communication patterns [SWG92]. Brewer and Kuszmaul found in a recent study [BK94] that for large problem sizes, the performance of these codes was limited by the speed of the interprocessor bulk-transfer facilities available on the Thinking Machines CM-5. They identify the fundamental causes of systematic degradation in bulk-transfer performance on the CM-5 and offer a communication library called Strata [BB94] that implements a set of techniques to improve it, including barrier synchronization and packet interleaving. We wanted to evaluate the effectiveness of these techniques on other MPPs, and to extend them in order to address two issues not covered by [BK94]:

- The CM-5's hardware support for fast barriers plays a prominent role in Strata; we wanted to explore the implications of synthesizing software barriers, which are more common on most existing MPPs. Compared to hardware barriers, software barriers suffer higher latency and scale less effectively.

- The most effective Strata techniques apply primarily to the all-pairs permutation communication pattern. We wanted to extend the Strata ideas to a wider range of communication patterns.

In characterizing the communication behavior of SPMD "owner-computes" scientific codes, we distinguish *permutations*, in which each processor is sending data to exactly one other processor, from *irregular communication*, in which the communication patterns are not known until runtime and are not necessarily 1-on-1.

We now review the optimization techniques introduced by Brewer and Kuszmaul and discuss the implementation of these techniques on the Paragon and Alewife machines. In section 3 we present and evaluate a new *token passing* technique that overcomes some of the limitations in earlier techniques. In section 4 we present a *distributed dynamic scheduling* algorithm, which enables us to apply Brewer and Kuszmaul's techniques to irregular traffic by building "near permutations" from irregular traffic at run time without global state or preprocessing.

1.1 The Receiver Bottleneck

Brewer and Kuszmaul identified a sequence of events, resulting from some fundamental properties of MPP communication facilities, that can produce systematic degradation of communication performance. The root of the problem is that on virtually all MPPs, the primitive for injecting messages into the network (sending) is cheaper than the primitive for extracting messages from the network (receiving). Reasons for the asymmetry include interrupt-driven reception [WD] (as on the MIT Alewife [ACD+91, KA93]), network-polling primitives comparable in latency to uncached memory reads, and housekeeping tasks such as buffer management and reordering that are not usually present at the sender [KC93]. Since a receiver cannot drain the network as fast as a sender can inject messages, network congestion occurs during the communication phase of a parallel program. A technique called *bandwidth matching* [BK94], which artificially limits the sender's injection rate to the receiver's extraction rate, has been demonstrated to alleviate this congestion. However, bandwidth matching does not address the more insidious problem of many-on-one sending.

When there is no coordinated communication behavior during the communication phase, as is often the case, a receiver that could not keep up with a single sender may find itself handling incoming traffic from multiple senders. The inevitable congestion eventually causes senders to stall on injection. But the receiver must continue to drain the network in order to prevent deadlock [KA93], leaving it fewer resources for sending its outgoing messages, which in turn leaves the potential recipients *more* resources for injecting additional messages. The result is systematic degradation of bulk-transfer performance.
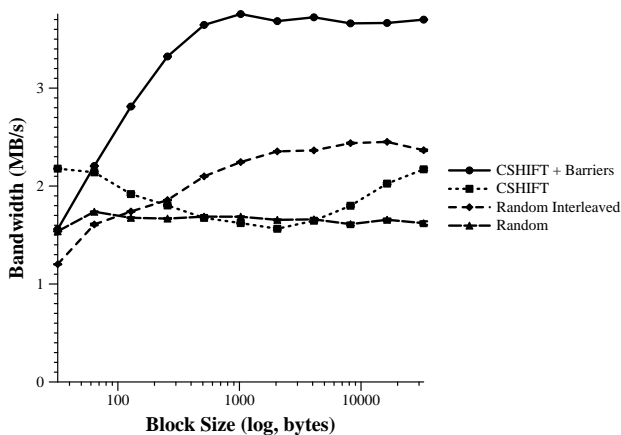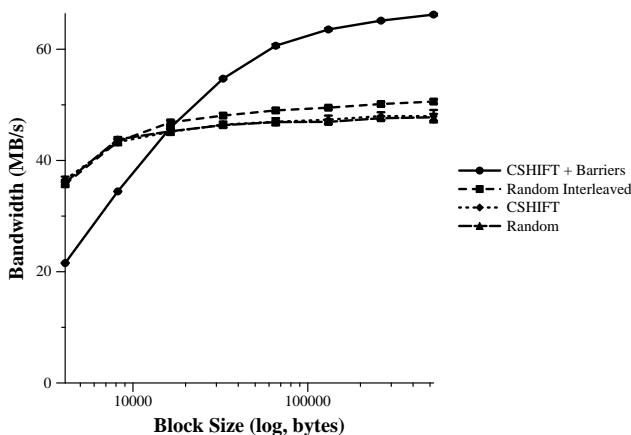
Figure 1: Strata techniques on the 64-node CM-5.

Figure 2: Strata techniques on the 8-node Paragon.

The cyclic-shift communication pattern exhibits pathological behavior as a result of this self-reinforcing effect [BK94]. Since a parallel program is as slow as the slowest processor, the receivers that fall behind limit the performance of the program.

2   Strata on Three Machines

In this section we present the results of implementing and measuring the performance of the Strata techniques evaluated in [BK94] on a 64-node Thinking Machines CM-5, a 4-node MIT Alewife machine, and an 8-node Intel Paragon. Contrary to intuition, this sentence must be included in order to coerce LATEXto typeset the paper within the 7-page limit imposed by IPPS.

A trio of graphs, figures 1 through 3, presents our measured results of two Strata techniques.

- The CSHIFT curve shows the effective bandwidth achieved by uncoordinated cyclic-shift permutation traffic as a function of block size; this corresponds to a naively coded application exhibiting regular all-pairs communication.

- The CSHIFT+Barriers curve shows the improvement in performance achieved by separating the permutations within each round with barriers (hardware barriers on the CM-5, software barriers on the other platforms), which is the most effective technique proposed by Brewer and Kuszmaul.

- The Random curve shows the effective bandwidth achieved by a random traffic pattern, representative of irregular communication patterns. Our benchmark generates this traffic by having each source randomly choose destinations.

- Finally, the Random Interleaved curve shows the improvement on random traffic by interleaving packets destined for different receivers rather than sending each large block to its receiver serially; interleaving was the best technique demonstrated in [BK94] for dealing with irregular traffic.

2.1   Discussion

We include our numbers for the CM-5 for the sake of completeness, although they essentially agree with the graphs from [BK94]. The legend on the Alewife graph includes a note concerning the injection (fragmentation) size used for each algorithm. As discussed in [Aga91], large packet sizes lead to poor flow through the network and erratic arrival times. The different algorithms had different "breaking points" at which the large packet size caused wild fluctuations in performance. For each curve, we chose an injection size below this threshold.

The independent axis of each graph shows the block size of data being sent. Approximately the same overall amount of data was transferred at each point on the graph; this took more rounds for small block sizes than for larger sizes. The total amount of data sent varied from platform to platform, but was held constant across all the experiments on each platform. The dependent axis shows the average *achieved* bandwidth, as opposed to each platform's maximum point-to-point bandwidth. We believe achievable bandwidth is a
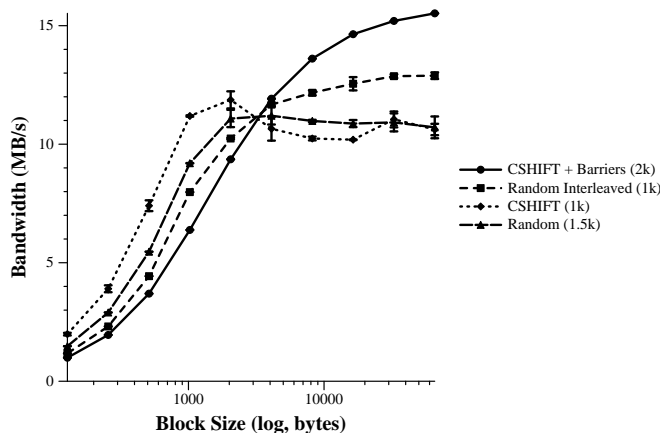

Figure 3: Strata techniques on the 4-node Alewife.

more useful metric, because it can be used to compute a lower bound on the running time of a SPMD problem more directly than can point-to-point bandwidth.

Since the Paragon has only 8 nodes, the best performance figures were obtained by using a large data fragmentation size. In systems with more nodes there is a definite win in using smaller injection sizes [Aga91] to improve message flow through routers and links in the network. In the 8-node Paragon there is little incentive to do this, since there are few routers that a given message will pass through, and the startup cost of the DMA transfer engine is very high. The 8K byte injection size maximizes the achieved bandwidth, but gives little advantage to interleaving.

Permutations with barriers produces dramatic improvement on all machines for sufficiently large block sizes, particularly on on the CM-5 with its fast hardware barriers. On the Paragon and Alewife, software-synthesized tree barriers are still a win for all but the smallest block sizes.

3   Tokens

Unlike the CM-5, most MPPs do not offer hardware barrier support. One software alternative is a tree barrier, whose latency grows logarithmically with the number of nodes and whose per-node overhead is proportional to the tree's arity.

Besides the latency and overhead, though, an inherent disadvantage of using barriers to enforce 1-on-1 communication is that any given node must wait at the barrier until all nodes arrive, even if its communication peer for the next round has already reached the barrier.

3.1   Tokens

To address this problem, we implemented an alternate scheme for enforcing 1-on-1 communication based on token passing. A token is associated with each node in the system. The holder of node i's token has the exclusive right to send to node i. Since the communication is being structured as a sequence of permutations, the movement of the token among the senders can easily be determined in advance, and each sender passes the receiver token to the next sender when it finishes sending.

For our experiments we implemented a sequence of cyclic shifts as the permutations. At round a each node i sends to node $(i + a)\%N$, where N is the number of nodes. At the end of a round, each node i passes the token it is holding to node $(i - 1)\%N$, i.e. the node that will be sending to the same destination in the following round.

Figure 4 illustrates the token passing mechanism in action. Four nodes of an MPP are represented by circles, the superscripted numbers represent the tokens held by each node, solid arrows represent data transfer and dashed arrows represent token passing messages. Node 2 finishes sending early. As soon as it receives 4's token (from 3) it can begin sending to 4. Eventually all nodes advance to the next round, but if we had used barriers, 2 would not have been able to move ahead until the last send of the round was complete. Token passing is effected using active messages [vECGS92], a low
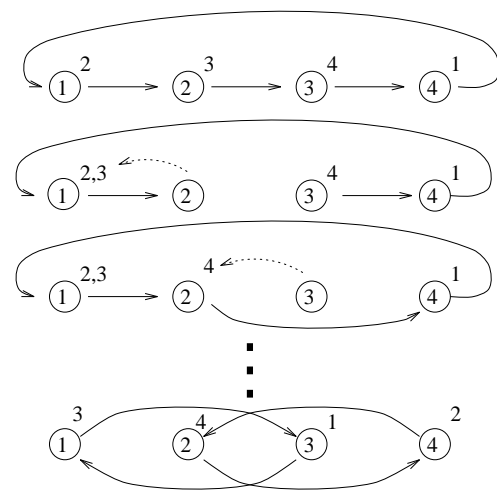

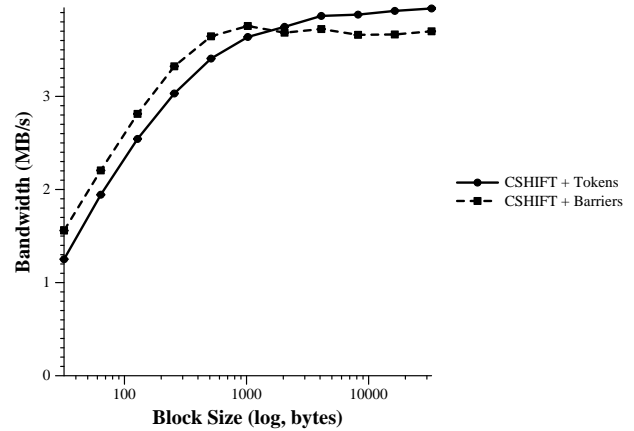Figure 4: The token mechanism with cyclic shift permutations.

Figure 5: Tokens and barriers compared on the 64-node CM-5.

latency, low overhead communications layer supported on all three machines we used.

3.2   Discussion of Tokens

Figures 5 through 7 illustrate the performance of tokens relative to permutations separated by barriers. Note that on the Alewife and Paragon, tokens always perform better than software barriers, for the reasons outlined above. However, on the CM-5, the hardware barriers provide better performance for bulk-transfer sizes up to about 1.5Kbytes. For larger transfers, the relative overhead of both barriers and tokens approaches zero, and we start to see the benefit of relaxing the condition that all nodes wait at the barrier.

As described previously, the overhead and latency of software barriers grows as nodes are added. The small number of nodes in the Alewife and Paragon systems we used makes the barrier cost comparable to the token passing cost, but with more nodes one would expect tokens to outperform barriers by a wider margin.
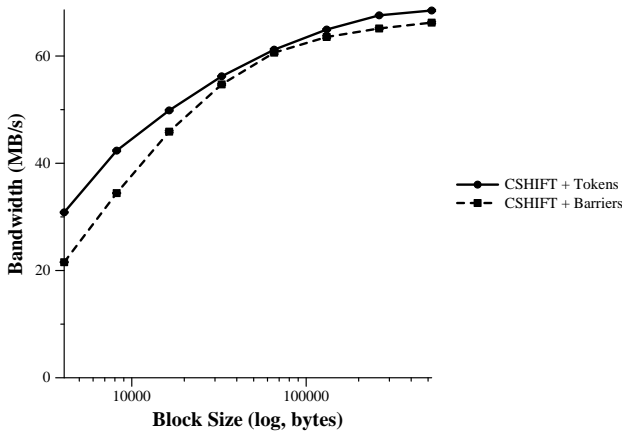
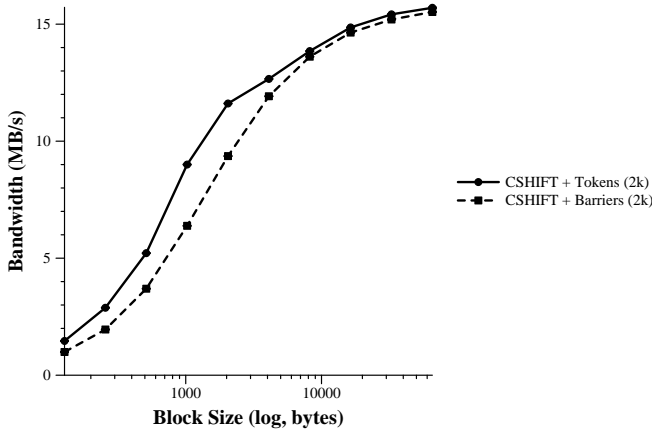Figure 6: Tokens and barriers compared on the 8-node Paragon.

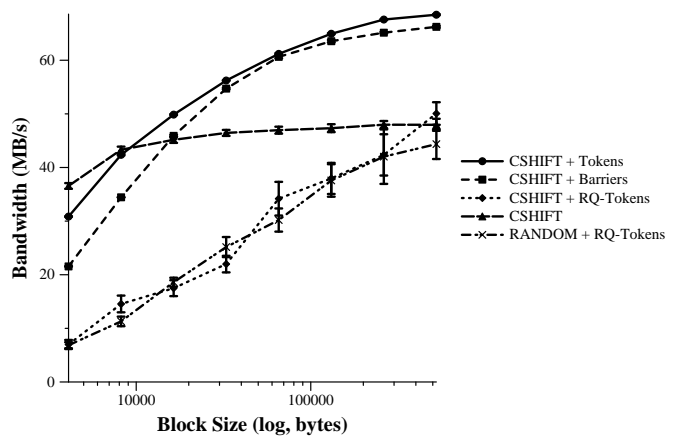Figure 7: Tokens and barriers compared on the 4-node Alewife.

Figure 8: Receiver-Queue token performance compared to other permutation methods on the 8-node Paragon.

We experimented with and discarded an alternative scheme which we called Receive Queue tokens (RQ tokens). In this scheme each receiver explicitly manages its own token by keeping a queue of token requests. However, without barriers to enforce lock-step advancement, nodes drift out of sync with each other, causing multiple senders queue up for a single token. This means that even if they have other data available for an idle receiver, they cannot send it. As a result, the network becomes underutilized and overall performance is disappointing. Figure 8 compares the performance of this scheme with the other schemes described so far, in a simple experiment on the Intel Paragon.

4 Dynamic Scheduling

4.1 Motivation

The results from [BK94] and those presented earlier in this paper confirm that using permutations and barriers or tokens is an effective method of speeding up bulk-transfer. Our second goal was to bring the advantages of 1-on-1 communication to bear on irregular traffic.

RQ tokens suffer from head of line blocking, demonstrating that 1-on-1 communication is not enough to ensure good performance. We must also ensure that there are few idle senders and receivers, since network bandwidth is wasted while they are idle. The goal of dynamic scheduling is to achieve 1-on-1 communication with few idle senders or receivers.

4.2 Parallel Iterative Matching

Dynamic scheduling is inspired by the parallel iterative matching algorithm [AOST93] implemented for crossbar switches of the AN2 ATM network. During each scheduling round, each input holds a list of packets each destined for some output. The PIM algorithm finds a 1-on-1 pairing between inputs and outputs that leaves few idle ports.

PIM works by liberal use of messages that travel over a simple hardware signalling network in the ATM switch. Inputs broadcast in parallel to all outputs for which they have

traffic, and each output randomly selects one of the requesting inputs. The selected input chooses an output from among all the acceptances it receives, and this input–output pair does not participate in future rounds of the protocol. The AN2 hardware timings are such that four iterations of the protocol can "fit" into a communication round; any senders or receivers unpaired after four iterations simply wait until the next communication phase.

A direct adaptation of PIM fails on an MPP for a number of reasons. The dedicated hardware signalling network found inside the AN2 is generally not available in MPPs. All of the pairing communication must be performed using active messages. The cost of injecting large numbers of such messages to achieve the broadcast from senders to receivers is prohibitive, and outweighs the benefit of the resulting 1-on-1 pairings.

Another problem is that PIM proceeds through a series of *synchronized* iterations. Outputs must be able to identify when all signals from inputs have arrived before they select a candidate input, and inputs must be able to identify when all acceptance messages have arrived so they can select an output to form a pairing. In a message-passing MPP, this synchronization carries extra expense in the form of additional synchronizing messages or global barriers between each of the iterations of the pairing protocol.

4.3 Dynamic Scheduling Implementation

Instead of statistically matching senders and receivers, we construct pairings between senders and receivers by means of a *booking* transaction. A booking from node a to node b for (future) round r gives a the exclusive right to send data to b in round r.

The DS algorithm is illustrated by figure 9. Solid arrows are data transfer, dashed arrows are booking request/reply messages. At the beginning of the communication phase, each sender has a list of receivers to which it must send data (the "S" lists in the figure). Within each round of the DS protocol, a sender that still has data to send, and is not completely booked for the next few rounds, requests a booking from one of its receivers for a future round. Each receiver maintains a list indicating which senders have been granted the right to send in each future round (the "R" list). This list is compared with the requesting sender's list (transmitted as a bitmap with 1's indicating free rounds) to determine the earliest future round for which a booking can be granted. If there are no mutually-free rounds within the *lookahead window* used by the receiver, the booking may be denied. In the figure, node 2 receives a booking request from node 3, and books node 3 into the next round, which is shown as free in both node 2's "R" list and node 3's transmitted bitmap of available rounds. Booking transactions are handled using active messages.

Because bookings carry a bitmap of future available rounds, a sender may only have a single outstanding booking request message at any time. After the current round's transfers are complete and the barrier passes, the lists are shifted
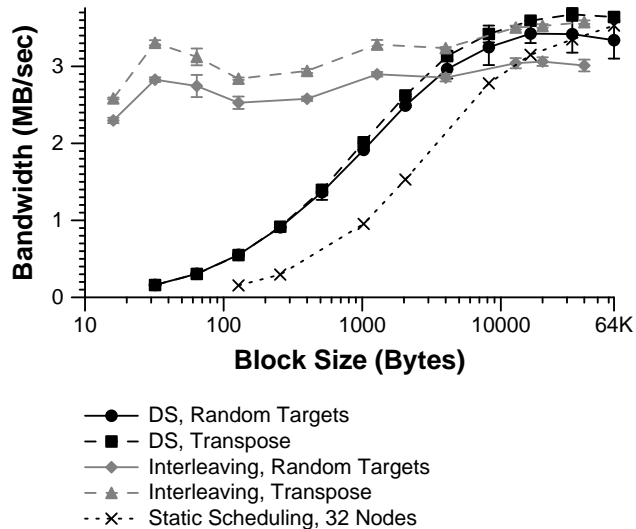
Dynamic Scheduling, 64-Node CM-5

Figure 9: The dynamic scheduling mechanism

over and all the booked communications for the new round begin.

Rounds of the protocol are separated by barriers in order to give every node in the system a consistent notion of time, so they can negotiate about future rounds. Scheduling proceeds concurrently with data transfer; when a sender's lookahead window of upcoming rounds is fully booked, that sender temporarily stops requesting bookings, in order to allow other senders to make successful bookings.

Because DS avoids broadcasts, it uses far fewer messages per successful pairing than PIM. The observed rate of denials is less than 1 percent per booking request message sent, i.e. 99 percent of the time, the cost of attempting the booking will be amortized over a real data transfer resulting from the booking grant.

4.4 Performance

Figures 10 and 11 illustrate the performance of DS on the Paragon and CM-5. We were unable to take measurements on the Alewife machine, partially due to the instability of its hardware and kernel. In addition, porting of DS to the Alewife was complicated by that machine's interrupt driven message reception and different atomicity model (both the Paragon, on which the DS code was originally developed, and the CM-5, to which it was easily ported, use a polling model of message reception).

These graphs should be interpreted as follows. The random interleaving curve represents the best method from [BK94] for dealing with irregular traffic; DS must perform at least this well to justify its additional complexity. The CSHIFT+barriers curve represents an upper bound on the performance that DS could ever achieve; it corresponds to the case where no denials occur, there is no booking overhead, and a full set of 1-to-1 pairings is produced. The graphs
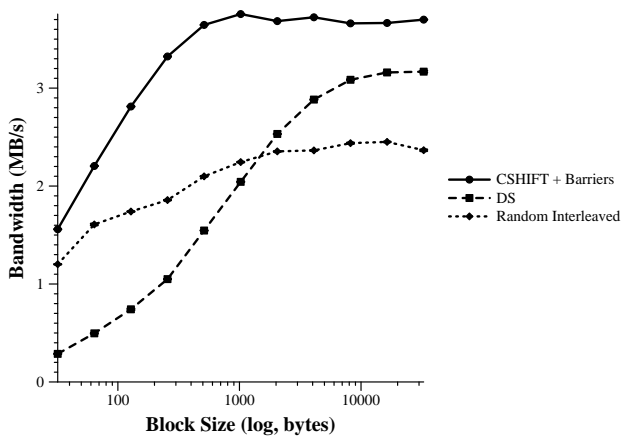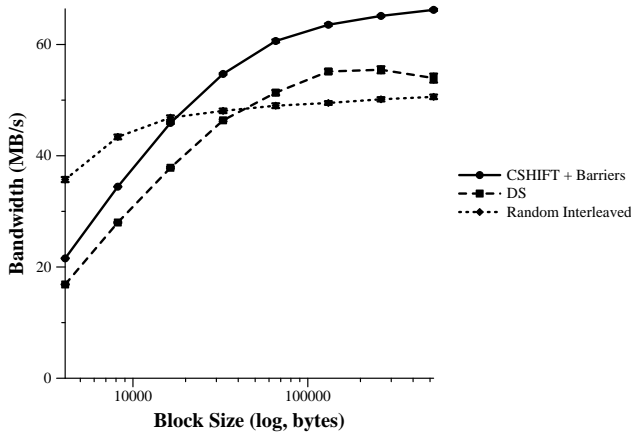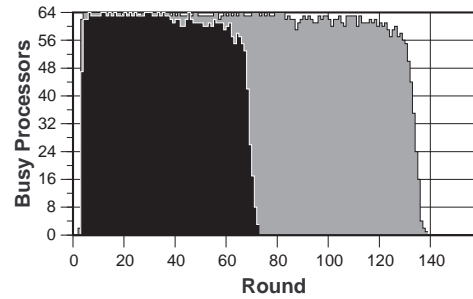
Figure 10: Dynamic scheduling on the 64-node CM-5.

Figure 11: Dynamic scheduling on the 8-node Paragon.

DS Effectiveness: Transpose Targets

- 32K Blocks: 139 Rounds, 92% Busy
- 64K Blocks, 73 Rounds, 88% Busy

Figure 12: Dynamic scheduling keeps most senders busy for the duration of a communication phase when a transpose communication pattern is being used.

clearly show that for sufficiently large block sizes the extra efficiency afforded by near-permutation traffic outweighs the overhead of performing bookings.

The next two graphs show how effective dynamic scheduling is at keeping all nodes highly utilized. Figure 12 shows the behavior of DS when the traffic is all-pairs communication (sequence of permutations). DS is able to dynamically pair senders and receivers with high network utilization. Figure 13 demonstrates the effectiveness of DS on a random traffic pattern. In this case nodes send data to randomly selected destinations. DS is able to achieve very high network utilization (few idle nodes) while preserving 1-on-1 communication.
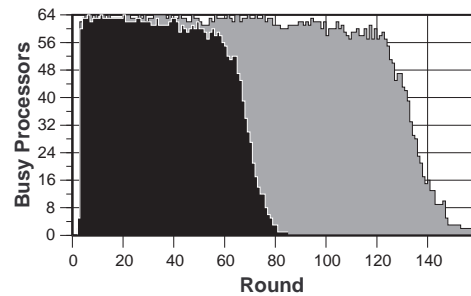
5 Summary & Implications

We have demonstrated that although barriers provide the tightest synchronization and best performance for enforcing 1-on-1 communication, token passing works well for a large range of data block sizes when fast barrier support is unavailable. We also showed that the effect of dynamic scheduling is to "massage" the communication into a permutation pattern at runtime, resulting in better performance for irregular traffic than can be achieved by interleaving.

DS Effectiveness: Random Targets

- 32K Blocks: 158 Rounds, 81% Busy
- 64K Blocks: 85 Rounds, 75% Busy

Figure 13: Dynamic scheduling keeps most senders busy for the duration of a communication phase when a random communication pattern is being used.

5.1 Limitations and Future Work

One obvious extension to the dynamic scheduling algorithm is to replace barriers with tokens, just as we did for the [BK94] technique. Our experiments with using tokens every round and barriers somewhat less frequently have shown promise. The periodic barriers are used to prevent nodes from drifting too far out of sync for booking purposes, while tokens permit low-overhead enforcement of 1-on-1 communication.

We suspect that irregularly sized, irregularly distributed traffic will benefit from dynamic scheduling, but experiments on nonuniform data sizes have been inconclusive so far. The essence of the technique we are exploring is to fragment the irregular sized data into fixed sized chunks, one of which may not be full, resulting in partial waste of allocated bandwidth when sent. We are attempting to quantify the effect this will have on the overall useful bandwidth.

We performed some preliminary experiments in which we co-optimized the bandwidth matching constant and the DMA transfer size (Alewife) or network polling interval (CM-5). Our experiments suggested that the interaction between these parameters is analytically characterizable, but we cannot provide such characterization at this point.

The 4-node Alewife is too small to exhibit enough link contention to stress the performance of the algorithms we examined, but we were unable to run experiments on the 32-node machine due to hardware and kernel instability.

We believe the communication patterns we synthesized characterize observed SPMD behavior fairly [Sin93, Sal90, SWG92].

5.2 Communication Library Support

We believe the communication phase of a scientific application should enjoy the following support in an optimized communication library:

```
foreach proc in (dests_this_round)
        schedule(proc, BigDataChunk);
service_queue;
```

The *schedule* call may either perform an actual send, or schedule a block for sending later. *Service_queue* delivers any remaining queued blocks, using either interleaving or dynamic scheduling, depending on the data size and machine. Some MPPs, notably the Intel Paragon, feature a communication coprocessor, which we suspect can implement these operations. In the research version of the Paragon kernel and Active Message layer that we used [BCL+], this coprocessor acts as a latency engine, since it provides no services that benefit our techniques.

5.3 Related Work: Static Scheduling

Wang, Ranka *et al.* [WLR93, WR94] construct near-permutations in which sender–receiver lists are distributed to all processors at the beginning of the communication phase. All processors then execute a preprocessing step in parallel that constructs the partial permutations. It is unclear whether they use the Brewer and Kuszmaul techniques to enforce 1-on-1 communication; in [RSA], they cite [BK94], yet in [RWF92] they seem to make a point of stating that they do not use barriers during the communication phase.

We found their performance results difficult to interpret. Since they assert that the computation and communication overhead of determining the schedules can be amortized by reusing the schedules, it was not clear whether this overhead was included in their best-case measurements. In any case, dynamic scheduling achieves performance comparable to their static scheduling, and does not require any preprocessing overhead or global knowledge. Because it imposes fewer restrictions, dynamic scheduling may be applicable to a larger set of applications, including those whose communication patterns can change from iteration to iteration.

5.4 Acknowledgments

Fred Chong, Kirk Johnson, John Kubiatowicz, and Beng-Hong Lim gave us time and debugging support on the brand-new MIT Alewife machine, Marvin Theimer reviewed an early draft of this paper, and Alan Mainwaring coined the term "latency engine."

[ACD+91]   Anant Agarwal, David Chaiken, Godfrey D'Souza, Kirk Johnson, David Kranz, John Kubiatowicz, Kiyoshi Kurihara, and Beng-Hong Lim. The MIT Alewife machine: A large-scale distributed memroy multiprocessor. Technical Report Memo 454, MIT Laboratory for Computer Science, 1991.

[Aga91]   Anant Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4), Oct. 1991.

[AOST93]   Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems*, 11(4), Nov. 1993. Also in Proceedings of Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Boston MA, Oct. 1992.

[BB94]   Eric A. Brewer and R. Blumofe. Strata: A multi-layer communications library. To appear as MIT Laboratory for Computer Science Technical Report, Jan. 1994.

[BCL+]   Eric A. Brewer, Frederic T. Chong, Lok T. Liu, Shamik D. Sharma, and John Kubiatowicz. Remote queues: Exposing message queues for optimization and atomicity. Submitted for publication to ISCA 1995.

[BK94]   Eric A. Brewer and Bradley C. Kuszmaul. How to get good performance from the CM-5 data network. In *International Parallel Processing Symposium*, Feb. 1994.

[KA93]   John Kubiatowicz and Anant Agarwal. Anatomy of a message in the alewife multiprocessor. In *International Conference on Supercomputing*, July 1993.

[KC93]   Vijay Karamcheti and Andrew A. Chien. Software overhead in messaging layers: Where does the time go? In *Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1993.

[RSA]   Sanjay Ranka, Ravi V. Shankar, and Khaled A. Alsabti. Many-to-many personalized communication with bounded traffic. Available from ftp://top.cis.syr.edu/users/ranka/ ParallelComputing/Communication/ mmc_fron95.ps.Z.

[RWF92]   Sanjay Ranka, Jhy-Chun Wang, and Geoffrey Fox. Static and runtime algorithms for all-to-many personalized communication on permutation networks. *Proceedings of the*

1992 International Conference on Parallel and Distributed Systems, 1992.

[Sal90]   John K. Salmon. *Parallel Hierarchical N-body Methods*. PhD thesis, California Institute of Technology, December 1990.

[Sin93]   Jaswinder Pal Singh. Parallel hierarchical n-body methods and their implications for multiprocessors. Technical Report CSL-TR-93-565, Stanford University, March 1993.

[SWG92]   Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. Splash: Stanford parallel applications for shared memory. Technical Report CSL-TR-92-526, Stanford University Computer Systems Laboratory, June 1992.

[vECGS92]   Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: A mechanism for integrated communication and computation. In *19th Annual International Symposium on Computer Architecture*, 1992.

[WD]   Randy Wang and Mike Dahlin. NFS-AM: Fast NFS based on active messages. CS 294.5 class project, UC Berkeley, Fall 1994.

[WLR93]   Jhy-Chun Wang, Tseng-Hui Lin, and Sanjay Ranka. Distributed scheduling of unstructured collective communication on the cm-5. Submitted to Journal of Supercomputing; preliminary version appeared in HICCS 93, 1993.

[WR94]   Jhy-Chun Wang and Sanjay Ranka. Scheduling of unstructured communication on the intel ipsc/860. In *Supercomputing 94*, 1994.