In Proceedings of ISCA'95, Santa Margherita, Italy, June 22-24, 1995.

A Comparison of Architectural Support for Messaging in the TMC CM-5 and the Cray T3D

Vijay Karamcheti and Andrew A. Chien
Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue, Urbana, IL 61801
{*vijayk,achien*}@*cs.uiuc.edu*

Abstract

Programming models based on messaging continue to be an important programming model for parallel machines. Messaging costs are strongly influenced by a machine's network interface architecture. We examine the impact of architectural support for messaging in two machines – the TMC CM-5 and the Cray T3D – by exploring the design and performance of several messaging implementations. The additional features in the T3D support remote operations: memory access, fetch-and-increment, atomic swaps, and prefetch.

Experiments on the CM-5 show that requiring processor involvement for message reception can increase the communication overheads from 60% to 300% for moderate variations in computation grain size at the destination. In contrast, the T3D hardware for remote operations decouples message reception from processor activity, producing high-performance messaging independent of computation grain size or variability.

In addition, hardware support for a shared address space in the T3D can be used to solve the output contention problem (output hot spots), producing messaging implementations that are robust over a wide variety of traffic patterns. Atomic swap hardware can be used to build a distributed message queue, enabling a "pull" messaging scheme where the destination requests data transfer upon receive. This scheme uses prefetches to mask receive latency. While this yields performance robust over output contention, its base cost is competitive only for small messages (up to 64 bytes) because of the high cost of issuing and resolving prefetches in the T3D. Emulation shows that if the interaction costs can be reduced by a factor of eight (250ns to 31ns), perhaps by moving the prefetch queue on chip, and there is a corresponding increase in the prefetch queue size, the pull scheme can give superior performance in all cases.

1 Introduction

In the past decade, parallel machines [8, 22] have become significant competitors to vector machines in the quest for highest performance computing. Applications on parallel machines rely on inter-node communication to transfer data, synchronize, and coordinate the parallel computation. This coordination determines the parallelism effectively exploited and hence the achievable machine performance. Communication performance depends critically on the network interface design, but if high-volume microprocessors are to be used, the network interface is often off chip, increasing the cost of processor-network interaction. This is the case for both the T3D and the CM-5, causing the network interface architecture to have a large impact on communication cost and performance.

Rather than focus on network fabric performance, our primary interest is to explore how the network interfaces affect communication performance. Thus, our experiments explore different interactions between the processor and network interface as well as traffic patterns that differ primarily in the output characteristics. In all cases, the network fabrics are more than sufficient to support the traffic – performance is limited by the network endpoints.

We compare the network interface hardware in two commercial machines, the Thinking Machines CM-5 [22] and the Cray T3D [8], examining how the network interface architecture affects the structure and performance of the software messaging layer.[1] In addition to using point-to-point benchmarks, we use more complex multi-party communication patterns, exploring network behavior for a range of patterns of use. This approach reflects our experience and that of others [2, 9] that network performance under aggregate loads is often quite different from unloaded scenarios.

We compare the communication performance characteristics of the Thinking Machines CM-5 and the Cray T3D, using traditional "push" messaging implementations which transfer data eagerly from source to destination. Because the processor mediates all network transfers in the CM-5, communication performance is linked to processor responsiveness (computation granularity). Slight delays in removing messages drastically reduce network performance even under essentially uniform traffic. For example, our experiments show that a variability of 50% of the average grain size produces a 60% increase in communication cost. As a result, it is difficult to achieve good performance on the CM-5 for computations or computational models that produce irregular grain size. Such programs either suffer poor communication performance or high polling overhead. This is perhaps a consequence of the fact that the CM-5 was designed to support a large base of data parallel programs where communication phases are synchronized – computation and communication phases do not overlap.

The Cray T3D dedicates hardware for remote memory operations, decoupling data transfer from local computation. Remote load/store and fetch-and-increment operations can be performed without processor involvement and can be used to build a fetch-and-increment messaging layer. In this layer, messages are received and

[1]Note that although the T3D has hardware support for a shared address space, a major fraction of its workload uses PVM [13], so messaging performance is a significant issue on the T3D.

buffered in local memory without involvement of the receiving processor. Our experiments show that this decoupling eliminates the sensitivity of communication performance to computational grain size, making the T3D better able to support programming models that overlap computation and communication or generate irregular concurrency. However, as with the CM-5, this scheme still sends data to the destination eagerly, in cases with modest fan-in, output contention can be a serious problem.

The T3D's atomic-swap can be used to build a messaging layer which eliminates the output contention problem. Using the atomic-swap hardware to link messages into a distributed message queue, sending can go at the maximum swap rate with the receiving processor "pulling" messages from the sender's memory. Receives incur a round-trip latency, but the prefetch queue can be used to mask this latency. Atomic-swap messaging eliminates output contention because data is only moved when the receiver is ready to process it. The atomic swap messaging layer provides robust network performance under a variety of aggregate traffic loads; there is essentially no degradation from uniform traffic to an all-to-one hotspot. The one shortcoming of atomic-swap messaging is the cost of prefetch issue and resolution, making the zero network load cost competitive with the fetch-and-increment scheme only for small messages (< 64 bytes). Our experiments show that reducing this interaction cost, perhaps by integrating the prefetch queue on-chip, can make the atomic-swap messaging implementation superior in all cases.

Specific contributions of the paper include:

- High performance fetch-and-increment and atomic swap implementations for messaging on the T3D. Both implementations, available as the Fast Messages (FM) Library[2], are an order of magnitude faster than PVM and provide performance comparable to the SHMEM put/get library.

- Demonstration and empirical characterization of network performance sensitivity to variations in computation granularity. On the CM-5, computation grain size distributions with standard deviations equal to and twice the the mean produce 100% and 300% increases in send overhead respectively. The remote load/store and fetch-and-increment hardware eliminate this sensitivity on the T3D.

- Demonstration of a T3D atomic-swap messaging (pull-based) layer which gives robust performance over a range of output contention. Send overheads for even all-to-one traffic show an increase of only 1.5x over that for all-to-all traffic. This is dramatically better than the 15x increase for T3D fetch-and-increment messaging and 25x for CM-5 Active Messages.

- Evaluation of the benefits of reducing the T3D prefetch costs on atomic-swap messaging. The base cost of atomic swap messaging is superior in all cases if the prefetch costs are reduced by a factor of eight with a corresponding increase in the prefetch queue size.

The remainder of this paper is organized as follows. Section 2 describes the CM-5 and T3D architectural features relevant for messaging. Section 3 describes the messaging implementations. The performance impact of processor responsiveness is presented in Section 4, particularly evaluating the impact of hardware for remote memory accesses in the T3D. Section 5 explores network interface impact on communication performance using aggregate traffic loads. This section includes demonstrations of the fan-in problem, first showing how the atomic-swap scheme eliminates this, and second how the T3D hardware can be improved to produce even better messaging performance. Sections 6 and 7 present our perspective and describe related work, and finally Section 8 summarizes our results.

[2]FM is available from http://www-csag.cs.uiuc.edu and also runs on the Myrinet LAN hardware.

2 Architectural Support for Communication

Both the CM-5 and T3D provide user-level access to the network interface, an important prerequisite for efficient communication. Because of the technology differences between the machines, the clock speeds are dramatically different, so direct quantitative comparisons are difficult. However, each machine provides completely different communication hardware, so we focus on the qualitative performance differences.

2.1 TMC CM-5

The CM-5 is a distributed memory parallel machine in which the network interface (NI) hardware provides a memory-mapped interface to the network. This interface resides on the main memory bus, and is as shown in Figure 1. Packets are injected into the network by storing the destination node number and data arguments to the NI send buffer, and extracted by loading the packet data from the NI receive buffer. Network interface status is determined by reading NI status registers. Despite fairly tight integration (the NI is on the processor's memory bus), each access to the network interface requires several processor cycles. Further, both sending and receiving a message requires active participation of the processor[3].

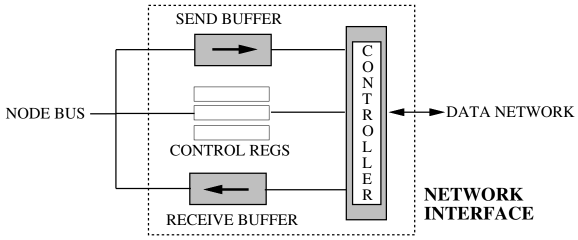

Figure 1: *The CM-5 network interface.*

The CM-5 architecture has three consequences for communication. First, the communication rate is limited to the rate at which the processor can move data across the memory bus. Second, communication operations occupy the processor, preventing the overlap of computation and communication. And finally, because the network requires consistent draining to maintain good performance [2], the processor needs to poll the network interface often. Since each poll requires a significant number of cycles, frequent polling can incur a significant overhead. The alternative is unattractive because the CM-5 network and NI contain only a modest amount of buffering, so infrequent polling backs up the network, reducing performance

[3]The CM-5 NI also supports an interrupt-driven interface but the processor must still extract the data from the network. Because interrupts are so expensive on the SPARC, we consider only a polled interface.

and jamming the network inputs. Such jamming causes send retries which can cost up to a microsecond per packet and may block the sending processor. These design choices reflect the expectation that inter-node communication is tightly synchronized, but have dramatic impact when it is not.

2.2 Cray T3D

The Cray T3D is a physically distributed memory machine with support for a shared address space. The DTB Annex allows a processor to read or write all of the machine's memory without the assistance of the remote processors. The T3D processing element structure is shown in Figure 2. The addressing of remote memory is not the same as local memory; it requires special operations in the DTB Annex. The T3D nodes include additional hardware for block transfers and message queues, but because of the associated overheads, that hardware is only relevant for large messages.

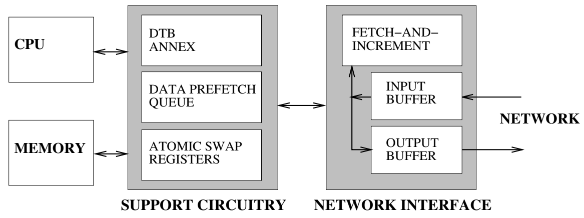

Figure 2: *The T3D processing element and network interface.*

T3D nodes also support three special memory operations for synchronization and latency hiding – fetch-and-increment, atomic swap, and prefetch [19]. First, each processor has a fetch-and-increment register that can be read or written by other processors. Reading the fetch-and-increment register atomically increments the register's contents and returns the original contents to the requesting processor. Second, each processor has a swaperand register which implements atomic swaps with remote memory locations. To swap, a program first sets up a remote memory location as a swap location then reads that location, causing the current value to be returned while atomically swapping the value of the swaperand register into the remote location. Third, processors can prefetch data; each prefetch returns a word to the 16-word prefetch queue. This is most useful for masking remote memory latency. These special operations all involve off-chip access, and thus are expensive (> 10 processor cycles).

Thus, the hardware support in the T3D differs from that in the CM-5 in two ways:

1. The T3D provides a variety of hardware primitives with which to build alternative messaging implementations.
2. Data movement in the T3D can be achieved in a one-sided fashion (involving only one processor)

In the next section, we describe messaging implementations on the CM-5 and the T3D which exploit specific hardware features to achieve high performance.

3 Messaging on the CM-5 and T3D

Messaging layers on parallel machines provide communication services to applications, implementing in software whatever is not provided directly by the underlying hardware. Typical features include reliable, in-order delivery of messages with flow control. We focus on minimal services – data transport and flow control – because they are fundamental to any messaging layer. Data transport is the essence of messaging and flow control (buffer management) ensures that there is space for messages at the destination. Flow control is required by general messaging models such as ones supported by PVM [13] or MPI [12], and programming models which produce concurrency dynamically [6, 4]. Note that primitives such as PUT/GET only specify data movement – flow control is still necessary, the source and destination processors must interact to decide where the message must be buffered. If there is no space, the destination must be able to block the source.

Programming models with dynamic concurrency also stress the communication system in other ways. Communication traffic in such systems is often characterized by irregular computation grain size, increased output imbalances (lots of messages destined for the same destination), and bursty traffic. A messaging layer must provide high bandwidth with low latency and overhead, and provide these services robustly under such workloads.

We consider the cost of implementing an active-messages like interface on both the CM5 and the T3D. In an active-messages interface, data transfers are coupled with invoking a handler at the destination node [24]. On the CM-5, this interface is implemented using the UC Berkeley active messages library (CMAM) [23]. On the T3D, we consider two implementations which comprise the Fast Messages (FM) Library – FM fetch-and-increment messaging, and FM atomic-swap messaging.

3.1 CM-5 Messaging

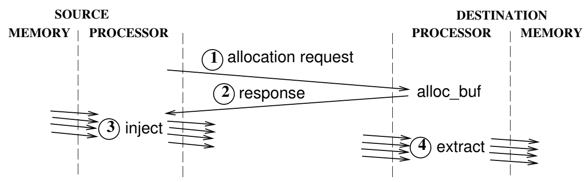

Figure 3: *CM-5 messaging implementation.*

Figure 3 shows the four steps in CMAM messaging. The source first sends a buffer allocation message to the destination (Step 1). On receiving a buffer request, the destination allocates a buffer of the appropriate size, associates it with a communication segment, and responds with the segment identifier (Step 2). The source then initiates the transfer by splitting the message into several CM-5 hardware packets, each of which carries the destination segment identifier (Step 3). The destination receives the data packets, reassembles them into the allocated buffer and invokes the specified user handler on transfer completion (Step 4). Flow control is ensured by the buffer allocation protocol; the destination does not respond until it has space to store the contents of the transfer. See [17] for a detailed breakdown of the software messaging costs in CMAM.

3.2 T3D Fetch-and-increment Messaging

The FM fetch-and-increment messaging primitives use the fetch-and-increment (F&I) hardware [19] as shown in Figure 4. The atomic increment of the F&I register is used to serialize message

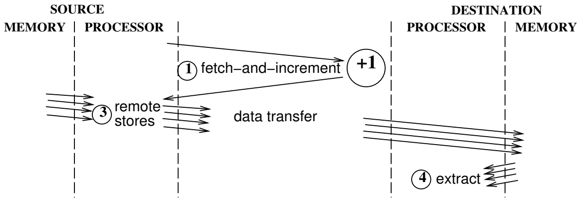

Figure 4: *T3D fetch-and-increment messaging implementation.*

reception and allocate buffers for incoming messages. The source increments the F&I register (Step 1) and on receiving an offset where the message should be stored, issues a set of remote stores (Step 2) to transfer the data to the destination buffer. Buffer overflow is detected by an invalid F&I value, requiring the source to retry the F&I operation. The destination is responsible for extracting messages from filled buffers (Step 3) and resetting the contents of the F&I register on detecting an overflow. This T3D implementation is similar to the CM-5 implementation in that data is eagerly moved to the destination node. This minimizes the latency of reception, but can cause contention at network outputs.

3.3 T3D Atomic-swap Messaging

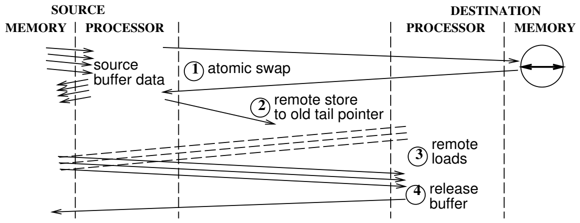

Figure 5: *T3D atomic-swap messaging implementation.*

The FM atomic-swap messaging primitives use the T3D swap hardware [19] to implement a distributed message queue. Each processor holds a tail pointer for its distributed message queue in a local memory location. Message transfer consists of four steps: First, the source writes its message data into a local buffer and swaps a pointer to the buffer with the contents of the tail pointer on the destination (Step 1). This makes the local buffer the last entry in the destination's distributed message queue. The source then stores a pointer to the buffer into the memory address swapped out of the tail pointer (Step 2). This operation links the previous end of the queue to the local buffer. To receive, the destination using remote loads to retrieve the message data (Step 3). On completion of the transfer it marks the source buffer as invalid (Step 4), permitting reuse of the buffer. There are no races between steps 1 and 2 at the source and step 3 at the destination, because the destination can detect the end of the distributed message queue and simply wait (or go and do some other work) until more messages have been added. Since Steps 1 and 3 require waiting for remote operations, the prefetch hardware is used to mask their latency.

This distributed queueing is a form of source buffering which avoids output contention (the fan-in problem). Sending to a node can happen at the full swapping rate, generally higher even than the memory bandwidth of the receiving node. Of course, the receive rate is limited by the capacity of the node to retrieve message data. One additional advantage is that buffer overflow happens rarely, only when a node has sent many messages that have not yet been consumed. This type of fan-out problem is much easier to solve than fan-in.

4 Single-source Single-receiver Messaging

In this section, we explore the performance impact of coupling the processor with message reception using a point-to-point communication workload. In the CM-5, message reception requires processor involvement but shared address-space hardware in the T3D decouples message reception from destination processor activity.

4.1 Experiments

The point-to-point messaging costs on the CM-5 and on the T3D (fetch-and-increment implementation) are examined by executing a synthetic kernel. As shown in Figure 6, this kernel alternates between computation and communication operations. Communication consists of the transfer of a 256 byte message (including buffer allocation at the destination), then a receive of any pending messages. To explore the impact of processor mediated reception, we first consider how increased compute time, then stochastic distributions of compute time with a specific mean and variance affect communication performance.

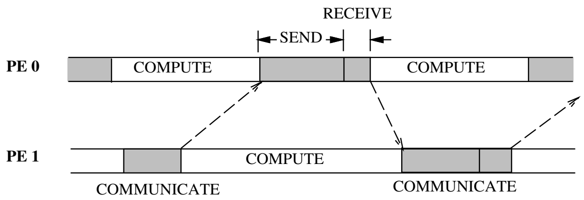

Figure 6: *Synthetic compute-communicate kernel used in the measurements. The shaded region corresponds to the communication overheads.*

All graphs and tables in this Section report the *communication overheads* for executing the compute-communicate kernel. This corresponds to the shaded region in Figure 6. Although many communication studies report message transfer latencies, overhead is a more relevant measure for computation models with latency-tolerance such as multithreading. In such systems, the important metric is how many processor cycles are spent on the communication operations and are therefore unavailable for computation.

Figure 7 shows the point-to-point send overheads in the CM-5 implementation as a function of both the mean compute time and its variance. The overheads include the cost of the buffer allocation request, and waiting for the response. When compute time is increased (with no variance), there is only a small increase in source overhead. This increase is due to the increased latency for responses to the buffer allocation request. However, as the granularity of the computation becomes irregular, communication performance suffers. As the compute time's standard deviation increases, communication performance decreases significantly. For example, when the standard deviation of the compute time is 50% of the mean, send overheads are 60% higher. When the standard deviation is twice the mean, send overheads are 300% higher. This

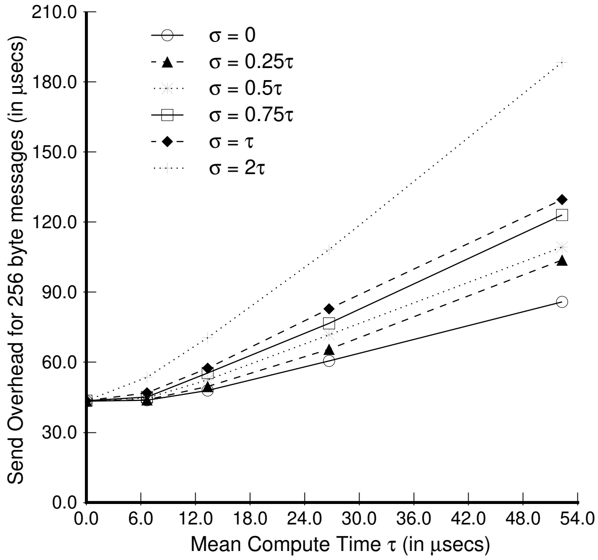

Figure 7: *Send overhead on the CM-5 as a function of mean compute time (τ) and its variance (σ). Send overheads increase by up to 60% when $\sigma = 0.5\tau$, and 300% when $\sigma = 2\tau$.*

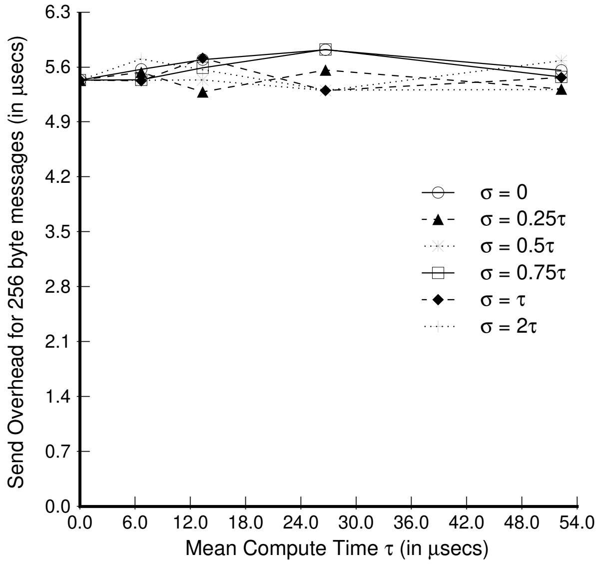

Figure 8: *T3D fetch-and-increment send overheads as a function of mean compute time (τ) and its variance (σ). Send overheads remain unaffected by variations in mean compute time.*

is because variability in compute time corresponds to delay in response to the network, and even small delays quickly back up the network. A congested network in turn holds up the sources, increasing source delay. Clearly, processor responsiveness is essential for good communication performance on the CM-5.

The Cray T3D provides special hardware for remote memory operations, allowing messages to be received without processor involvement and removed from the network without delay. Figure 8 shows the send overheads seen by the same compute-communicate kernel for the T3D fetch-and-increment implementation. Note that the absolute times for the same kernel are much lower, reflecting the higher basic speed of the T3D. As compute time increases, as with the CM-5, network performance remains good. In addition, when the standard deviation of the compute time is increased, network performance remains good. The additional hardware for remote requests decouples the local computation from the network, providing robust network performance.

Finally, we considered improving the overheads of the T3D fetch-and-increment implementation by using the prefetch hardware to overlap the fetch-and-increment with local computation. However, as Table 1 shows, little benefit can be achieved. This is because the overlapping scheme requires the buffering of message data while waiting for the response to the fetch-and-increment. The additional memory operations for buffering in addition to the cost of prefetch make this approach unattractive.

In summary, the results in this section show that processor responsiveness on the CM-5 is critical for performance since communication requires active processor involvement to extract packets from the network. In contrast, the T3D provides hardware primitives and features which eliminate the need for direct processor involvement, thereby providing robust performance irrespective of destination processor activity.

Messaging Implementation	*Message size (in bytes)*			
	16	64	256	1024
Fetch-and-incr	2.76	3.04	5.29	14.65
Fetch-and-incr (with prefetch)	2.42	3.42	6.92	19.68

Table 1: *Comparison of send overheads in the T3D fetch-and-increment messaging implementation, with and without prefetch of the remote operation.*

5 Irregular/Imbalanced Message Traffic

In this section, we consider the performance of the three messaging implementations under multi-party communication traffic. First we demonstrate the fragility of the CM-5's network performance under irregular or slightly imbalanced workloads. Second, we consider the overhead of the T3D fetch-and-increment implementation and show that despite better hardware support, performance still degrades under irregular workloads. The basic problem with both of these schemes is that they do not deal well with output contention because they optimistically "push" messages to their destination. Third, to solve the problem, we develop a "pull based" T3D atomic-swap implementation which is robust with respect to imbalances in destination distributions. The T3D hardware support for atomic swap and prefetch are essential to making this messaging implementation scheme competitive. Finally, we consider how the T3D's hardware primitives could be improved to make the atomic swap implementation's performance superior in all cases.

5.1 Irregular Traffic Patterns

In the previous section we considered point-to-point communication performance on an unloaded network. We now consider the following traffic distributions which load the network and generate varying output contention spanning from none to extreme.

1. *all-to-all*: processor i sends a message to processor $(i + 2) \bmod N$. This permutation pattern generates moderate network contention but no output contention (fan-in), and is used as a baseline to evaluate the effect of output contention.
2. *random*: each processor picks a random destination and sends to it. Since several processors may pick the same destination, the pattern generates moderate fan-in.
3. *hotspot* ($4\times$, $16\times$): the same as random except that one node is designated a hotspot (processor 0) and receives 4 times or 16 times the traffic of the other nodes. The hot spot intensity is proportional to the amount of fan-in.
4. *all-to-one*: all processors (except processor 0) send messages to processor 0. This represents the extreme hotspot situation where all the traffic is sent to a single processor.

For each traffic patterns, we measure the overheads for sending a fixed size message and receiving a queued message.[4] All processors execute a kernel which consists of a fixed size message send, a receive, and a computation part (modeled by a stochastic variable delay). In all cases, we measure the *steady state* send and receive overheads (defined as in Section 4.1) seen by the nodes. All graphs in this Section present the harmonic mean over all the nodes of these steady state overheads. *It should be clarified that we are not measuring the time to send and receive a fixed set of messages: we are measuring the overhead incurred per send and per receive given that the processors in the machine are generating and consuming traffic at steady state*. The experiments reported in this section all used 64-node machine configurations.

CM-5 Messaging Performance

Figure 9 shows CM-5 network performance for the traffic patterns for four message sizes. Note that this is a log-log scale. The point-to-point costs have been included for comparison. The three graphs in Figure 9 vary the compute time and its standard deviation.

For all message sizes, the overheads for all-to-all traffic are an order of magnitude larger than the point-to-point overheads, showing the impact of limited buffering (even with no output contention) in the CM-5 network. For example, for 1024 byte messages (compute time: $\tau = 0.3\mu$s, $\sigma = 0$), the point-to-point overheads are 100μs versus 980μs for the all-to-all pattern. More significantly, output contention produces dramatic degradations in performance, increasing source overheads by another order of magnitude from the all-to-all case. For the message size considered above, the source overheads are 9000μs for the hotspot,x16 pattern, and 21550μs for the all-to-one pattern. Our results demonstrate the classic "hotspot" saturation [20], where a single blocked output can back up and significantly degrade communication performance for even unrelated traffic. This degradation actually dwarfs the effects of processor non-responsiveness reported in the Section 4.1.

[4] The CM-5 Active Messages implementation actually extracts *all* pending messages on every send so as to clear the network on every occasion. However, for the traffic patterns considered, these additional receives have negligible impact on the source overheads, which are dominated by the time for send retries.

T3D Fetch-and-increment Performance

The performance of the T3D fetch-and-increment layer is shown in Figure 10 for a range of message sizes and compute time distributions. As before, the absolute performance is much better than in the CM-5, but this is largely due to hardware technology.

As expected, variations in compute time distribution have negligible impact on the results, indicated by approximately the same curves across all three graphs. Comparing source overheads for all-to-all traffic with the point-to-point case show the effect of increased buffering in the T3D network. For example, for 1024 byte messages (compute time: $\tau = 0.3\mu$s, $\sigma = 0$), the point-to-point overheads are 20μs versus 45μs for the all-to-all pattern. Despite better performance under network loading, the results show that the fetch-and-increment scheme is not robust over output contention. As the hotspot intensity increases (greater imbalance in message destinations or larger messages), source overheads increase by up to an order of magnitude as compared to the all-to-all case. For the message size considered above, the source overheads are 140μs for the hotspot,x16 pattern, and 685μs for the all-to-one pattern.

As in the CM-5 case, this performance loss is due to classic hot spot saturation which backs up the network, slowing the injection of all messages into the network. Such hot spot contention will occur in any messaging layer which eagerly pushes messages to their destination. Although such contention is increasingly critical in large-scale machines, it is difficult to detect and manage.

T3D Atomic-swap Performance

The atomic swap implementation can eliminate network output contention by enqueuing messages onto a distributed message queue and moving data to the destination only upon request. Figure 11 shows the performance of the swap-based messaging implementation for the various traffic patterns. The send overheads increase linearly with message size because of the source copy of the message. As with the fetch-and-increment case, variations in the compute time do not have any noticeable impact on the results.[5]

The atomic swap implementation exhibits robust performance irrespective of the traffic pattern as indicated by the close clustering of the curves in each graph. As the intensity of the hot spot increases, there is no noticeable degradation in performance. This is in stark contrast to the CM-5 and T3D fetch-and-increment performance which is quite sensitive to output contention. The major reason for atomic swap's robustness is the benefit of distributed queueing and lazy data movement. In the atomic-swap messaging scheme, hot spots elevate the swap rate at the destination; however, since the data is not transmitted, a single node can easily support extremely large swapping rates – supporting even all-to-one communication for extremely large machines with no degradation in performance. The hot node only "pulls" in message data as required, matching the data transmission rate to that feasible for the node. Thus, there is neither excessive data traffic or swap traffic to interfere with other communication. Further, because the swapping rate does not depend on message size, the atomic swap based scheme also scales with message size. In fact, we have been effectively unable to generate traffic loads which cause any input or output contention using atomic-swap messaging.

[5] A closer examination of the source overheads indicates that initially the overheads reduce with an increase in the compute time, but after a threshold they do not change: this reduction results from a better overlap of the prefetch operations with the node computation.

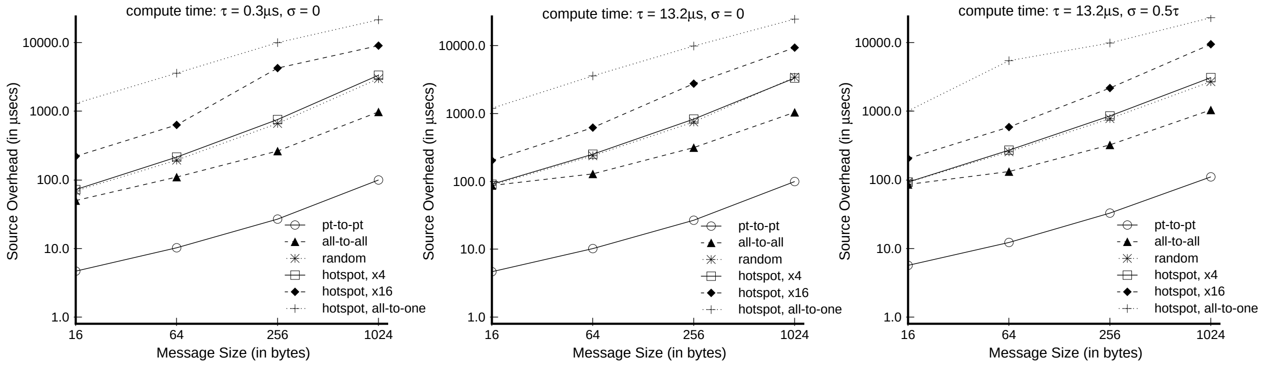

Figure 9: *CM-5 source overheads. Source overheads increase by up to 2 orders of magnitude over all-to-all traffic with increase in output contention.*

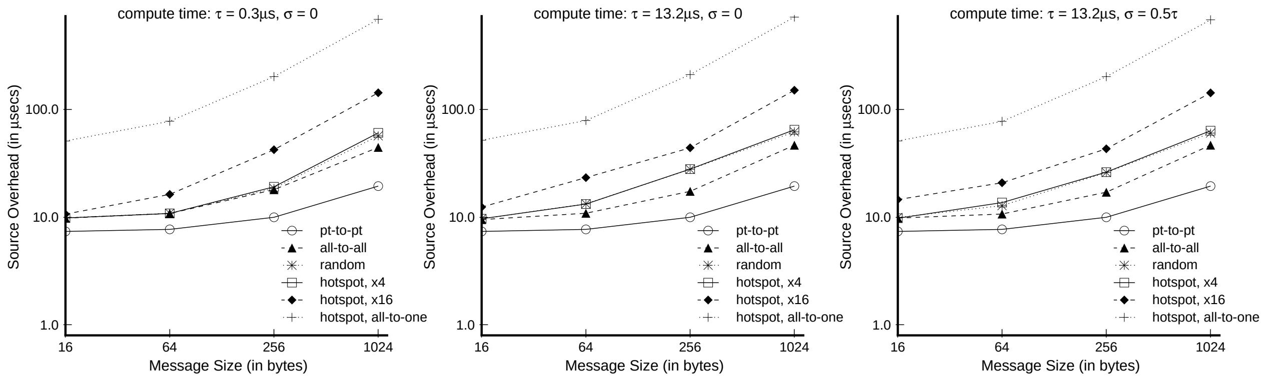

Figure 10: *T3D fetch-and-increment source overheads. Although absolute performance is better than for CM-5 messaging, source overheads still increase by up to an order of magnitude over the all-to-all case.*

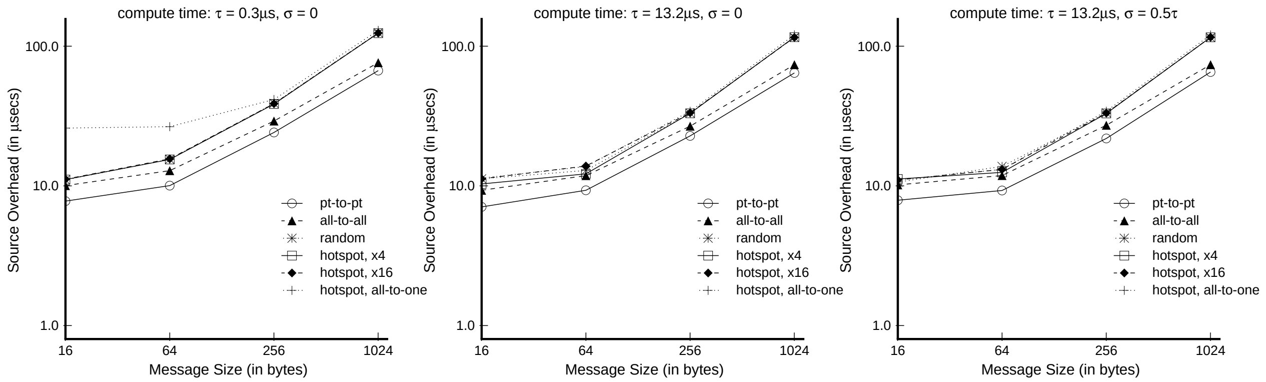

Figure 11: *T3D atomic-swap source overheads. Source overheads are insensitive to output contention, with overheads for all-to-one traffic being only 1.5x that for all-to-all traffic.*

Messaging Implementation		*Message size (in bytes)*						
		16	32	64	128	256	512	1024
Fetch-and-increment	Src	2.76	2.77	3.04	4.26	5.29	8.26	14.65
	Dest	1.08	1.08	1.08	1.08	1.08	1.08	1.08
Atomic Swap	Src	3.84	4.09	4.24	4.54	5.24	6.85	10.16
	Dest	5.82	8.06	12.36	20.66	36.74	68.12	130.07
Atomic Swap (with prefetch)	Src	2.24	2.34	2.51	2.90	3.73	6.19	9.47
(overlap = 5.50μsec)	Dest	1.95	2.55	3.69	8.23	14.29	25.75	45.27

Table 2: *Base source and destination overheads of the T3D Fetch-and-increment and Atomic-swap messaging schemes. (all times in μs).*

While the benefits of atomic swap and source-based queueing may not be surprising, demonstrating a high performance implementation of a "pull" based messaging implementation is significant. Absolute performance of the fetch-and-increment and atomic swap messaging is shown in Table 2. Source and destination overheads for the base atomic swap implementation (i.e., without prefetch) are also shown for comparison. Atomic swap is competitive for small messages, but pulling the message data incurs additional remote memory latencies which dominate for large messages. The prefetch hardware can be used to mask that latency, reducing the cost of both the swap and pulling the data. With the T3D prefetch hardware, atomic-swap is more competitive, but is significantly worse than fetch-and-increment messaging for messages larger than 64 bytes. The culprit here is the current implementation of prefetching in the T3D.

In the T3D, the overhead difference between the atomic-swap and fetch-and-increment implementations is largely due to the high cost of interaction with the prefetch queue: 0.1μs (15 cycles) to issue a single word fetch, and 0.15μs (20 cycles) to extract a word from the prefetch queue. Below, we explore possible enhancements of the prefetch hardware which further reduce the cost of the atomic swap scheme, making its performance competitive in all cases.

5.2 Architectural Improvements for Atomic Swap

The major source of additional overhead in the atomic swap scheme is is the cost of issuing and resolving prefetch operations. Table 3 shows the prefetch queue interaction costs for a range of message sizes as a fraction of the total atomic-swap messaging overhead. The high interaction cost is because the prefetch operations need to access off-chip hardware, nearly 0.25μs per fetch and extract pair. Integrating the prefetch queue on-chip would enable lower interaction costs and dramatically reduce this overhead.

Message size (in bytes)						
16	32	64	128	256	512	1024
0.30	0.36	0.44	0.48	0.49	0.52	0.60

Table 3: *Prefetch queue interaction costs as a fraction of the net atomic swap messaging overhead.*

To explore the impact of cheaper prefetch, we emulated an enhanced machine, reducing prefetch cost and increasing prefetch queue size in tandem. To preserve the T3D's balance, any reduction in prefetch cost should be matched with a corresponding increase in the prefetch queue size. Our emulation scheme is simple: to model the effect of a lower prefetch queue interaction cost and a larger prefetch queue, we "pull" only a fraction of the data sent at the source. This emulation allows a precise modeling of proportional decreases in prefetch cost and increases in queue size. In Table 4, we examine the impact of three levels of improved prefetch hardware – halving the interaction costs while increasing the queue size by a factor of two, reducing the interaction costs by a factor of four, while increasing the queue size by four times, and reducing the interaction costs by a factor of eight while increasing the queue size by eight times. The interaction costs shown in Table 4 are for issuing one fetch and extracting one word from the prefetch queue.

From the results in Table 4, the performance of the atomic swap scheme is competitive with the fetch-and-increment scheme when the interaction cost is reduced to 31.25 nsecs per word along with a prefetch queue size increase to 128 words. Integrating the prefetch queue on chip would likely produce a performance benefit of eight or even greater. Increasing the prefetch queue size by a factor of eight is technologically feasible, but may be difficult due to the fact that most microprocessors are primarily designed for use in sequential systems. However, perhaps performance studies such as this can convince designers of the importance of such features.

6 Discussion

While this study was performed in the context of two specific architectures, the CM-5 and the T3D, we believe the results are relevant to all parallel machines with high speed interconnects – both massively parallel machines as well as networks of workstations.

The design choices in the CM-5 architecture are perhaps a consequence of its focus on supporting a particular programming model – data parallel programming. However, there is a danger to designing machines this way because such designs are unable to cope with the communication requirements of more dynamic programming models needed for irregular computations. In data parallel models, computation and communication phases are synchronized and do not overlap. Since all processors cooperate in a communication phase, requiring processor involvement in message reception does not adversely affect performance. In contrast, dynamic programming models exhibit irregular traffic patterns (both in time and space) with increased output contention. Consequently, if each processor is working asynchronously, output non-responsiveness can severely restrict the achievable communication performance. These models either suffer from poor network performance or from excessive polling overhead to ensure that the network is serviced frequently. Thus, in the CM-5, output responsiveness is likely to be a more important contributor to performance than the network performance itself.

In contrast, although the T3D was designed as a shared address machine, its support for efficient messaging is considerably better than in the CM-5. In addition to data transfer primitives to directly

Emulation Parameters		Message size (in bytes)						
queue size	interaction cost	16	32	64	128	256	512	1024
16	250.00 ns	1.95	2.55	3.69	8.23	14.29	25.75	45.27
32	125.00 ns	1.48	1.77	2.50	6.17	7.05	10.99	22.44
64	62.50 ns	1.48	1.45	1.77	5.51	6.19	9.55	15.68
128	31.25 ns	1.47	1.44	1.44	3.72	4.86	5.71	9.63

Table 4: *Impact of changes in prefetch queue size and interaction costs on destination atomic swap messaging overhead (all times in μs).*

read from and write to another processor's memory, messaging still requires that the processors interact to agree on where message data must be placed. This buffer management step increases the cost of messaging over the base data transfer costs. The fetch-and-increment hardware enables buffer management and thereby allows the entire messaging operation to be implemented without involving the destination processor. This architectural feature of the T3D supports better responsiveness and network performance because draining the network is now independent of destination processor activity. Despite robust performance independent of destination processor activity, the fetch-and-increment scheme still suffers from poor performance for traffic patterns with output contention. This degradation is a characteristic of "push" messaging which causes contention at the output. Despite this, "push" messaging continues to be the traditional implementation in most existing parallel machines. This is as much a reflection of the limited communication demands of SPMD and data parallel programs as it is of a hardware characteristic which sustains them, since most machines do not provide architectural support to implement an alternative.

Architectural support such as the atomic swap on the T3D, coupled with a shared address space and prefetch hardware enables a distributed message queue implementation of messaging. This supports a "pull" model of messaging where the destination receives data at a rate that it can sustain. As we have shown in this study, such hardware support allows better performance characteristics and holds the possibility of robust performance for a wider range of programming models and traffic patterns. Although such an implementation is usable on the T3D today, it is competitive only for short messages. For longer messages, the prefetch costs cause performance to be worse than that of the fetch-and-increment scheme. Reducing the cost of prefetch by a factor of eight along with a matching increase in the prefetch queue size (128 words) can make the atomic swap based scheme provide higher performance over the entire range of message sizes. The atomic swap scheme demonstrates the flexibility of shared address space primitives in providing robust messaging performance. This contrasts with traditional attempts to improve messaging performance by providing specialized hardware support to integrate message processing within the processor. Such approaches are effective only for reducing the point-to-point messaging costs, but provide no solutions for network and output contention.

Although the T3D supports a shared address space, the addressing of remote and local data is not uniform. Remote data access requires an interaction with the DTB Annex – 0.3μs to setup a remote address – and consequently limits achievable performance. Since irregular communication patterns require frequent access to the annex (to set up an entry for the appropriate destination node), significant reductions in messaging costs are possible with a lower annex setup cost.

This study is a beginning, more work needs to be done to determine the right set of network interfaces which can provide robust messaging performance.

7 Related work

While several studies have measured communication performance on parallel machines [7, 21, 15], they have not explored the relationship between network interface architecture and messaging performance. A great deal of attention has also been focused separately on specialized hardware support for messaging, the design of efficient messaging layers, and network management for improving communication performance. The primary distinction of our work is that it spans all these areas, evaluating the hardware support and messaging protocols required to provide robust performance for a range of dynamic and irregular traffic patterns.

Research on specialized hardware support for messaging has focused primarily on integrating message processing within the processor [10, 14, 1, 25]. These approaches are effective in reducing point-to-point costs, but provide no solutions for network and output contention. In contrast, we have investigated messaging atop shared address space primitives and demonstrated that it can deliver performance robust over output contention.

Research on messaging layer approaches to improve communication costs have typically investigated minimizing kernel interaction [22, 11, 3] and the active messages [23] approach of offloading all but the essential operations from the messaging layer. Our study evaluates architectural support for messaging and shows that while the above approaches reduce the point-to-point messaging costs, they are inadequate to prevent performance degradation due to output contention.

Researchers have also explored ways of managing the network to improve communication performance. While most of this work deals with single link management [18], others [2] have looked at obtaining performance on the CM-5 data network for all-pairs communication traffic. However, their solutions (frequent barriers and packet reordering) consider only permutation traffic patterns with moderate output contention and require all processors to cooperate for achieving high performance. In contrast, we have examined messaging schemes under more irregular and dynamic traffic patterns which produce varying degrees of output contention.

8 Conclusion

We have evaluated the performance impact of architectural support for messaging in two parallel machines, the TMC CM-5 and the Cray T3D. In particular, we have examined the impact of hardware which frees the processor from message reception, hardware for remote fetch-and-increment and atomic swap, and also hardware for prefetching by considering the performance of several messaging implementations under a range of traffic patterns. Our study shows that the messaging implementation on the CM-5 is extremely sensitive to the granularity and variance of computation on the destination processor, showing overhead increases from 60% to 300% for moderate variations in the computation grain size. This sensitivity is not present in a T3D implementation which uses the fetch-and-increment hardware and remote store operations to free the destination processor from message reception. However, the CM-5 implementation and this T3D implementation, both "push" messaging implementations, suffer severely degraded performance in the face of output contention with communication overheads increasing by up to 2 orders of magnitude.

The atomic swap hardware, along with the prefetch hardware on the T3D permits a "pull" messaging implementation which is insensitive to output contention, and provides robust performance over a range of traffic patterns. However, although its cost is competitive for short messages, the high cost of interacting with the T3D prefetch queue results in higher base cost for longer messages. Emulation shows that reducing the prefetch costs by a factor of eight along with a matching increase in the prefetch queue size to 128 words makes the pull scheme competitive for all message sizes.

This study was motivated by the high communication costs that we observed in our CM-5 implementation of a concurrent object-oriented language which generates very irregular communication traffic [5, 16]. We are currently in the process of porting the implementation to the T3D. Future work will examine the implications of architectural support in the context of traffic patterns generated by large application programs.

Acknowledgements

The authors thank Jae Hoon Kim and John Plevyak for participating in several discussions about the messaging experiments described in this paper. We also thank the anonymous reviewers for their useful comments.

The research described in this paper was supported in part by NSF grants CCR-92-09336, MIP-92-23732 and CDA-94-01124, ONR grants N00014-92-J-1961 and N00014-93-1-1086, NASA grant NAG 1-613, and a special-purpose grant from the AT&T Foundation. Andrew Chien is supported in part by NSF Young Investigator Award CCR-94-57809. The experiments reported in this paper were conducted on the CM-5 at the National Center for Supercomputing Applications and the T3D at the Pittsburgh Supercomputing Center.

References

[1] S. Borkar, R. Cohn, G. Cox, T. Gross, H. T. Kung et al. Supporting systolic and memory communication in iWarp. In *Proceedings of the 17th International Symposium on Computer Architecture*, pages 70–81. IEEE Computer Society, 1990.

[2] Eric A. Brewer and Bradley C. Kuszmaul. How to get good performance from the CM-5 data network. In *Proceedings of the International Parallel Processing Symposium*, pages 858–867, 1994.

[3] H. Levy C. A. Thekkath and E. D. Lazowska. Separating data and control transfer in distributed operating systems. In *Proceedings of the Sixth Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, 1994.

[4] K. Mani Chandy and Carl Kesselman. Compositional C++: Compositional parallel programming. In *Fifth Workshop on Compilers and Languages for Parallel Computing*, New Haven, Connecticut, 1992.

[5] Andrew Chien, Vijay Karamcheti, and John Plevyak. The Concert system – compiler and runtime support for efficient fine-grained concurrent object-oriented programs. Technical Report UIUCDCS-R-93-1815, Department of Computer Science, University of Illinois, Urbana, Illinois, June 1993.

[6] Andrew A. Chien. *Concurrent Aggregates: Supporting Modularity in Massively-Parallel Programs*. MIT Press, Cambridge, MA, 1993.

[7] S. Chittor and R. Enbody. Performance evaluation of mesh-connected wormhole-routed networks for interprocessor communication in multicomputers. In *Proceedings of Supercomputing*, pages 647–56, 1990.

[8] Cray Research, Inc. *Cray T3D System Architecture Overview*, 1993.

[9] David Culler, Anurag Sah, Klaus Erik Schauser, Thorsten von Eicken, and John Wawrzynek. Fine-grain parallelism with minimal hardware support: A compiler-controlled threaded abstract machine. In *Fourth International Conference on Architectural Support for Programming Languages an Operating Systems*, pages 164–75, 1991.

[10] William J. Dally et al. The J-Machine: A fine-grain concurrent computer. In *Information Processing 89, Proceedings of the IFIP Congress*, pages 1147–1153, August 1989.

[11] Peter Druschel and Larry L. Peterson. Fbufs: A high-bandwidth cross-domain transfer facility. In *Proceedings of Fourteenth ACM Symposium on Operating Systems Principles*, pages 189–202. ACM SIGOPS, ACM Press, December 1993.

[12] Message Passing Interface Forum. The MPI message passing interface standard. Technical report, University of Tennessee, Knoxville, 1994.

[13] G. Geist and V. Sunderam. The PVM system: Supercomputer level concurrent computation on a heterogeneous network of workstations. In *Proceedings of the Sixth Distributed Memory Computers Conference*, pages 258–61, 1991.

[14] D. S. Henry and C. F. Joerg. A tightly-coupled processor-network interface. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages an Operating Systems*, pages 111–122, 1992.

[15] R. W. Hockney and E. A. Carmona. Comparison of communication on the Intel iPSC/860 and Touchstone Delta. *Parallel Computing*, (18):1067–1072, 1992.

[16] Vijay Karamcheti and Andrew Chien. Concert – efficient runtime support for concurrent object-oriented programming languages on stock hardware. In *Proceedings of Supercomputing'93*, 1993.

[17] Vijay Karamcheti and Andrew Chien. Software overhead in messaging layers: Where does the time go? In *Proceedings of the Sixth Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VI)*, 1994.

[18] R. Metcalfe and D. Boggs. Ethernet: Distributed packet-switching for local computer networks. *Communications of the Association for Computing Machinery*, 19(7):395–404, 1976.

[19] Robert W. Numrich. The Cray T3D address space and how to use it. Technical report, Cray Research, Inc., August 1994.

[20] G. F. Pfister and V. A. Norton. Hot spot contention and combining in multistage interconnection networks. *IEEE Transactions on Computers*, C-34(10):943–948, October 1985.

[21] R. Ponnusamy, R. Thakur, A. Choudhary, and G. Fox. Scheduling regular and irregular communication patterns on the CM-5. In *Supercomputing '92*, pages 394–402, 1992.

[22] Thinking Machines Corporation. *The Connection Machine CM-5 Technical Summary*, October 1991.

[23] T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: a mechanism for integrated communication and computation. In *Proceedings of the International Symposium on Computer Architecture*, 1992.

[24] Thorsten von Eicken and David E. Culler. *Building Communication Paradigms with the CM-5 Active Message layer (CMAM)*. University of California, Berkeley, 2.4 edition, September 1992.

[25] Colin Whitby-Strevens. The transputer. In *Proceedings of 12th International Symposium on Computer Architecture*, 1985.