Challenging Applications on Fast Networks

Koen Langendoen Rutger Hofman Henri Bal
Department of Mathematics and Computer Science
Vrije Universiteit, Amsterdam, The Netherlands
{koen,rutger,bal}@cs.vu.nl

Abstract

Parallel computing on clusters of workstations is attractive because of the low costs in comparison to MPPs, but the speed of the local area network limits the class of applications that can be run efficiently. Fortunately, faster network technology is becoming available for the next generation of workstation clusters. This paper studies the effect of running challenging applications that communicate heavily on three types of modern interconnects: 100 Mbit/s Fast Ethernet, 155 Mbit/s ATM, and 1.28 Gbit/s Myrinet. Experimental results show that even challenging communication-intensive applications can achieve acceptable performance on workstation clusters, but only if the communication software has been designed and tuned for high performance.

1. Introduction

Parallel computing on clusters of workstations is attractive because of the low costs in comparison to commercial Massively Parallel Processors (MPPs). A disadvantage of workstation clusters is that MPPs contain high-performance proprietary interconnection networks that are much faster than the typical Ethernet local area network. This performance gap, however, is decreasing because better network technology for interconnecting workstation clusters is becoming readily available (e.g., ATM [18], Myrinet [4]). The availability of such fast networks allows for the construction of workstation clusters that rival the performance of MPPs.

On the other hand, adding a high-speed network to a workstation cluster increases the system price, so this approach is only beneficial if the fast network significantly increases the performance of parallel applications running on the cluster. A key issue thus is the expected performance improvement for applications. In this paper, we will describe a quantitative study using several applications on two high-speed networks: ATM and Myrinet. These two networks were chosen because of their impact on cluster computing and because they are used in many research projects.

The question we address in this paper is how much performance improvement is obtained for parallel applications if a high-speed network like ATM or Myrinet is used instead of a traditional LAN. To obtain meaningful quantitative results, we use a modern, low-cost LAN for the performance comparison: 100 Mbit/s Fast Ethernet. An important aspect of our comparative study is that we use identical, state-of-the-art base hardware for all three networks (200 MHz Pentium Pros with 64 Mbyte of memory). In addition, we run as much as possible the same software on all three networks.

We use a variety of realistic parallel applications for our study. We deliberately selected only "challenging" applications. Applications that are easy to parallelize and do little communication will perform just as well on a slow network, so they will obviously benefit very little from a fast network. Most applications in our suite obtain an efficiency below 50% (on 8 nodes) when executed over Fast Ethernet and below 25% when using 10 Mbit/s Ethernet. The applications were selected from several application domains, and use widely different communication patterns. Some applications are latency-bound and send many small messages, while others are throughput-bound and send large amounts of data.

The main contribution of the paper is a performance analysis of a number of challenging applications on Fast Ethernet, ATM, and Myrinet. We also describe the performance of the three networks for low-level communication primitives and communication kernels, and analyze the performance behavior of applications in terms of these lower-level benchmarks using a simple performance model.

2. Experimental setup

In this section we describe the experimental setup used for our performance study. An important aspect of this research is that we use exactly the same base hardware (processors, caches, and memory) for all experiments. Unfortunately this is not the case for the software. The main difference is that Fast Ethernet is accessed through the kernel, while ATM and Myrinet are accessed from user space.

2.1. Hardware

Our system is a homogeneous cluster of Pentium Pro processors. Each node contains a 200 Mhz Pentium Pro (on a motherboard with a PCI bus), 64 MByte of EDO-RAM, and a 2.5 Gbyte IDE disk. Each Pentium Pro processor has an 8 KByte L1 instruction cache, an 8 KByte L1 data cache, and a 256 KByte unified instruction/data L2-cache.

The entire cluster consists of 64 nodes, but only 8 of these nodes are equipped with all three network technologies used for this paper. For our experiments, we thus use 8 nodes, all of which have three network connections: Fast Ethernet, ATM, and Myrinet. The Fast Ethernet network uses SMC EtherPower 10/100 adaptors and a KTI Networks KF1016TX hub. The ATM network uses ForeRunner PCA-200E interfaces and one Fore ASX200-WG switch. The Myrinet network uses Myricom SAN technology, consisting of LANai-4.1 interfaces (M2M-PCI32C) connected by 8-port switches (M2M-DUAL-SW8).

An important difference between the networks concerns reliability. The error rates for Myrinet are extremely low, so it can be considered a reliable network [13] (i.e., the hardware does not drop or corrupt messages). Fast Ethernet and ATM, on the other hand, can and do drop messages, so these networks provide unreliable communication. With all three networks, the interface boards reside on the PCI bus. The hardware bandwidth of the networks are 100 Mbit/s for Fast Ethernet, 155 Mbit/s (full duplex) for ATM, and 1.28 Gbit/s (full duplex) for Myrinet. ATM and Myrinet use a switched topology. The 8 ATM interfaces are connected to a single switch. For Myrinet, we use a subset of total network consisting of two 8-port switches that are connected to each other, and to four hosts each. Since FastEthernet is primarily used for file server traffic, it uses a hub, not a switch, to reduce cost.

2.2. Software

The overall software structure of our experimental system is shown in Figure 1. Applications are written in PVM and Orca [3], which are implemented on all three networks using a portability layer called Panda [14]. The application code and the PVM and Orca libraries are identical at the binary level: all differences are in the Panda layer and downwards. Panda provides threads, point-to-point communication, Remote Procedure Call, and totally-ordered group communication (multicast). Panda is implemented on top of Illinois Fast Messages (for Myrinet and ATM) or UDP/IP (for Fast Ethernet).

FM acts as a device-independent interface for Myrinet and ATM. In both cases, the network interface is accessed directly in user space to avoid the overhead of crossing the user/kernel boundary. FM does not enforce protection and thus allows only one user at a time. For parallel programming, however, it usually is desirable not to share the CPU with other users anyway, so this is not a serious drawback.

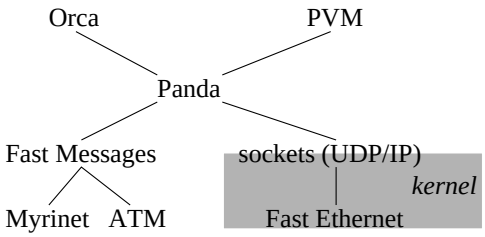


Figure 1. Software structure.

An advantage of accessing network interfaces from user space is that is much easier to adapt and optimize communication protocols. For Myrinet, we have customized the firmware for the programmable processor (LANai) on the interface board. We have extended the LANai Control Program (LCP, version 1.1) of Fast Messages (FM) [13] with a reliable and efficient spanning-tree multicast protocol [17]. This protocol runs entirely on the network interfaces, so it forwards multicast messages without involving the host processors. Another extension to the LCP is an efficient primitive to get a sequence number, which is used by our totally-ordered multicast protocol (described later). This primitive sends a message to a centralized sequencer machine; the network interface firmware on that machine replies with the next sequence number. For ATM we were not able to customize the firmware, since no no tools nor documentation was made available by Fore.

For Fast Ethernet, in contrast to Myrinet and ATM, we use the kernel-based UDP/IP protocol, and run Panda on top of it (in user space). With the current BSD/OS operating system it is neither possible, nor realistic, to access Fast Ethernet in user-space; the OS uses Fast Ethernet internally (e.g., to fetch executables from a file server) and is not willing to share the device with untrusted code in user space. As a consequence, the performance differences we will report are 'end-to-end' differences, caused partially by differences in the network technology and partially by differences in software organization.

2.2.1 Low-level software for Myrinet and ATM

The Fast Messages layer provides a standard interface for both ATM and Myrinet. This interface is similar to Active Messages. For Myrinet, the FM layer does message fragmentation and reassembly (using fixed-size packets of 256 bytes), thus allowing Panda to send messages of arbitrary size. The Myrinet FM layer also does flow control for point-to-point messages, to avoid losing messages due to buffer overflow [13]. As a result, that layer provides reliable communication (neither the network nor the communication

software drop messages). On ATM, however, the FM layer onely provides a uniform interface to the ATM driver. The ATM FM layer does not implement flow control or reliability, and supports packet sizes up to 4K. The Panda layer for ATM provides a protocol for reliability, fragmentation, and flow control.

2.2.2 Panda

The goal of the Panda system is to implement high-level communication primitives and threads on top of whatever primitives are provided by the underlying system. Internally, Panda has a highly modular structure [14]. Panda can be configured statically to use the protocols that best match the underlying system. For example, it only uses a reliability protocol if the communication primitives of the underlying system are unreliable, as is the case for Fast Ethernet and ATM. Each communication module delivers its messages by making upcalls to the layer above.

On systems that already provide reliable message passing (Myrinet), Panda's point-to-point messages and Remote Procedure Call are easy to implement. On systems with unreliable message passing (Fast Ethernet and ATM), Panda uses a credit-based reliability protocol, which also performs fragmentation and reassembly. Credit updates also serve as acknowledgements, so flow control and reliability are integrated. A time-out mechanism is used for retransmissions.

Besides point-to-point communication, Panda also supports totally ordered multicast, which guarantees that all multicast messages are received by all destinations in the same order. These strong semantics are useful for implementing distributed shared memory systems; it is used extensively in the Orca system.

The multicast protocol uses a centralized *sequencer* machine to order all messages. If the underlying system already provides reliable multicast, the protocol is simple: the sender first gets the next sequence number from the sequencer and then multicasts the message with the sequence number tagged on it. We run this protocol on Myrinet using our firmware extensions.

On systems with unreliable multicast (i.e., ATM and Fast Ethernet), we use another sequencer-based protocol, which roots in Kaashoek's work [9]. For small messages, a totally ordered multicast takes only one point-to-point message and one multicast by the sequencer. For large messages, a range of sequence numbers for all fragments of the message is requested from the sequencer, and the fragments are broadcast by the originator of the message. The ordering overhead in this case is just two control messages.

Finally, an important issue that needs to be discussed is how to receive messages when accessing network interfaces in user space (i.e., on Myrinet and ATM). Many user-level communication systems receive messages by polling the network adapter. While polling avoids the overhead of interrupt-based mechanisms, it is not suited for all parallel applications. Panda therefore uses a combined approach that uses both polling and interrupts to receive messages. Users need not insert polling statements into their code; through a careful integration of the user-level communication software with the user-level thread scheduler, Panda can automatically switch between polling and interrupts. Panda has been shown to achieve robust performance: in most cases, it performs as well as or better than systems that rely exclusively on interrupts or polling [10].

2.2.3 Orca

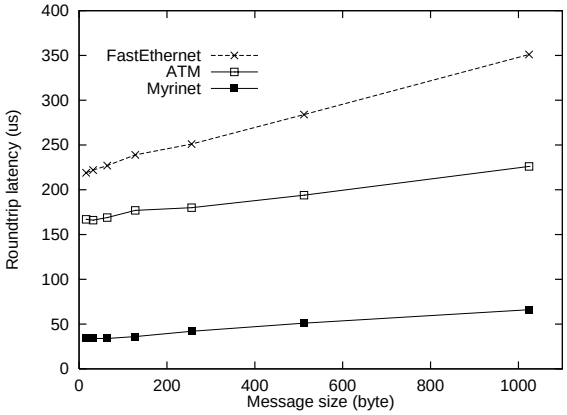
Orca is a language for writing parallel applications that run on distributed memory systems. Its programming model is a form of object-based Distributed Shared Memory [3]. Objects in Orca are variables of abstract data types that can be shared by different processes.

Orca is implemented on top of Panda as follows. An Orca process is implemented as a Panda thread. All Orca processes on the same machine thus run in the same address space. An Orca object is stored either on one machine or it is replicated on all machines. The Orca compiler and runtime system together decide (transparently to the programmer) which objects will be replicated. In general, objects that are read frequently and modified infrequently are replicated, to decrease communication overhead.

An operation on a nonreplicated remote object is implemented using a Remote Procedure Call, which requests the RTS on the remote machine to execute the operation. For operations on replicated objects, we use a different strategy. If the operation does not modify the object, it is executed locally, without any communication. If, however, the operation does modify the object, the operation and its parameters are sent to all machines containing a replica, using totally-ordered multicast. All these machines execute the operation on their local copy, thus updating the replica. The total ordering guarantees that all machines receive all updates in the same order, so all replicas will remain consistent.

2.2.4 PVM

We ported a subset of the PVM implementation for Solaris (version 3.3.8) to our Pentium Pro cluster, using the Panda portability layer [14]. We omitted PVM's dynamic process support and heterogeneity. Porting PVM programs to the Panda-based PVM system typically only requires a few changes to the initialization code. The original PVM system uses *daemon processes* for all communication, which introduced an unacceptable overhead. Instead, we implemented all functionality in a library rather than using an additional process, similar to the *direct links* supported by later PVM enhancements.



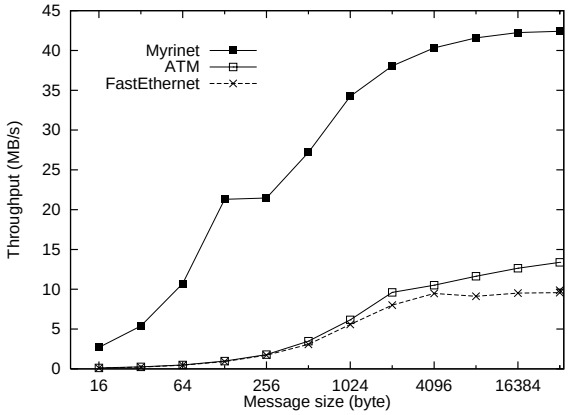


Figure 2. Panda message passing performance.

PVM applications receive messages by calling a (blocking or nonblocking) receive primitive. In these explicit receive calls, the user can specify both the sender and the *tag* of the message it wants to receive. Panda, on the other hand, uses an upcall model for receiving messages. We implemented the PVM model on top of Panda's upcall model by receiving all messages asynchronously via upcalls. The upcall handler stores the message it just received in a queue. The PVM receive call checks if the queue contains messages that match the specified sender and message tag.

3. Low-level communication

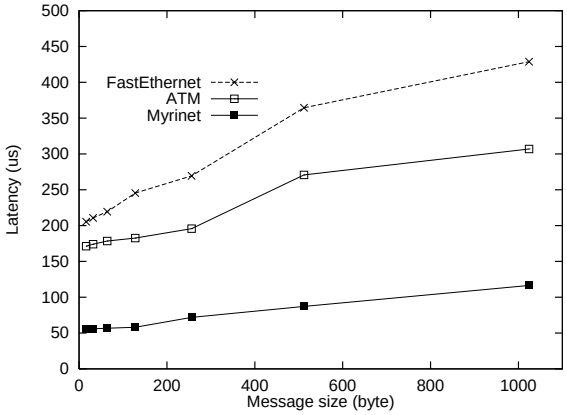
Although the goal of the paper is to study the performance of applications on fast networks, we first present the base performance numbers of Panda communication primitives on our hardware. These base numbers are essential for understanding the behavior of applications. Also, the base communication performance indicates how the three different fast networks compare to each other. Recall, however, that Panda had to be implemented differently on each network to offer the same functionality, so performance differences may not be attributed to the network hardware alone.

Figure 2 shows the performance of Panda's message passing primitives on the three fast networks. The roundtrip latency is obtained by measuring the elapsed time of performing 10,000 RPCs with a request of the specified size and an empty reply message between two nodes in the Pentium Pro cluster, and computing the average time. All messages are handled through polling, since the test program is always waiting for the next message. In practice an application can be busy computing when a message arrives. In that case an interrupt will occur to handle the incoming message, which adds 20 μs to the roundtrip latency reported in Figure 2. The throughput numbers are measured by having one node send messages of the specified size in a tight loop, while the other node just consumes.

The performance results show that Panda's message passing primitives implemented on top of Myrinet outperform the other two implementations by far. The roundtrip latency of a null message on Myrinet is only 34 μs, which is much lower than that for ATM (167 μs) and Fast Ethernet (219 μs). There are several causes for this large difference. First, the Myrinet hardware is faster than that of ATM and Fast Ethernet. For example, crossing a Myrinet crossbar switch takes less than 1 μs, while a message routed through the ATM switch incurs a 10-15 μs delay. Also, Myrinet transmits signals much faster (at 1.28 Gbit/s) than ATM (155 Mbit/s) and Fast Ethernet (100 Mbit/s). Second, the firmware driving the Myrinet network interface has been aggressively optimized for obtaining speed; the Fast Messages software outperforms the original Myricom API by more than a factor of three [12].

A third performance winner is that Myrinet provides reliable communication at the hardware level. This removes the need for a reliability protocol in software as implemented by Panda on ATM and Fast Ethernet. We measured that adding reliability, fragmentation and flow control to the basic Panda protocol on Myrinet adds 6 μs (17%), so having reliable network hardware is an advantage. The main cost of a software scheme for reliability is in the management of sequence numbers and credits.

The main difference between the performance on ATM and Fast Ethernet is caused by the BSD/OS kernel, which is used by the Panda implementation on Fast Ethernet. First, the kernel uses a generic UDP/IP protocol that is apparently not optimized for low latency. Second, using the kernel implies system call overhead to send and receive socket data. Third, the kernel enforces an additional copy of the message (from message buffer to socket buffer) before it can be sent out on the network; the FM interface used by Myrinet and ATM directly sends the data from the message buffer. We also noticed that the UDP/IP implementation in BSD/OS does not include any sophisticated buffer management, so



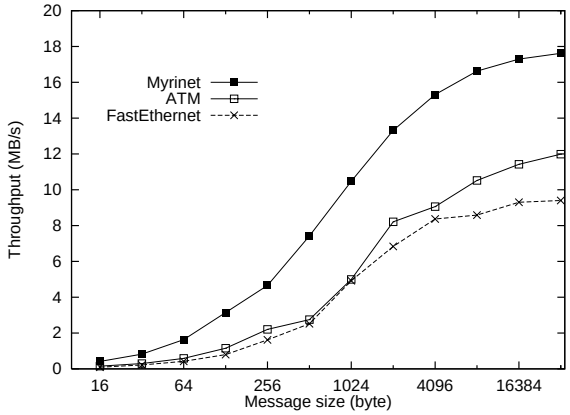


Figure 3. Panda group performance.

packets get lost at the receiver side as well as at the sender side if the sender puts data into the socket too rapidly. This is solved by running a flow-control protocol in Panda that limits the amount of outstanding data with a narrow sliding window.

The throughput curves in Figure 2 show that ATM and Fast Ethernet approach their respective link speeds (155 Mbit/s and 100 Mbit/s) for large message sizes. Myrinet, on the other hand, while achieving a much higher throughput (42 MB/s = 336 Mbit/s), is well below the 1.28 Gbit/s speed of the Myrinet links. We found that the bottleneck is the Myrinet LANai Control Program that needs to handle each incoming (256 byte) packet. The LCP runs on a 40 MHz custom processor, which is too slow to keep up with the 200 MHz Pentium Pro processor. Increasing the packet size to 512 bytes will improve throughput, but will also increase latency for small messages since FM uses a fixed packet length. In addition, the knee in the Myrinet curve at 256 bytes, caused by fragmentation/reassembly overhead, will shift to the right.

Figure 3 shows the performance of the Panda group protocol. The latency is obtained by measuring the elapsed time of sending 1000 messages of the specified size back and forth between all pairs of 8 nodes using group communication (i.e., multicast), and then computing the average time to send one group message. All messages are handled through polling, and the interrupt overhead (20 μs) should be added in the case of an application that computes when a group message arrives. The throughput numbers displayed in Figure 3 are measured by having each host in turn send 10,000 messages of the specified size in a tight loop. Note that the throughput numbers are *sender* throughput; if the receiver throughput is desired, all numbers must be multiplied by 7 ($= |\text{processors}| - 1$). These throughput numbers represent the maximum bandwidth available from one sender only in the group. Performance numbers for multiple sending group members will be described in Section 4.

The group latency numbers show the same trend as the message passing numbers: Panda on Myrinet is much faster than ATM, which is faster than Fast Ethernet. This is rather surprising since both Fast Ethernet and ATM provide hardware support for multicast while Myrinet does not. The excellent performance of Panda on Myrinet is caused by our FM LCP extensions that provide a spanning-tree multicast protocol running directly on the network interfaces (see section 2.2), and has support for obtaining sequence numbers at the firmware level. To send a group message on ATM or Fast Ethernet, at least one extra message must be sent from user space to user space to implement the total ordering.

The throughput curves in Figure 3 show that Panda on Fast Ethernet and ATM obtains slightly lower bandwidth than the message passing protocol for large messages. On Myrinet, however, the Panda group protocol only achieves half the throughput of the message passing protocol. We presume that the forwarding delays on the network interfaces are the throughput bottleneck.

The possibility to modify the firmware running the Myrinet network interfaces is a big advantage: we have measured that multicast protocol implemented by the firmware is 14% faster for null messages than a similar protocol running solely on the host machines; the maximum throughput is 33% higher.

4. Communication kernels

Although the low-level performance numbers for message passing and group communication provide some insight in the behavior of Panda on the three fast networks, these numbers are not sufficient to explain application behavior when network congestion comes into play or when communication between different hosts is done in parallel. Therefore we have studied (and optimized) several communication kernels (or patterns) that occur in our benchmark applications. We report on our experience with the follow-

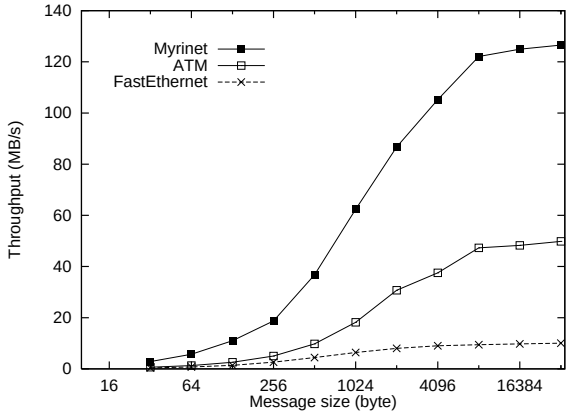


Figure 4. ring throughput (on 8 nodes).

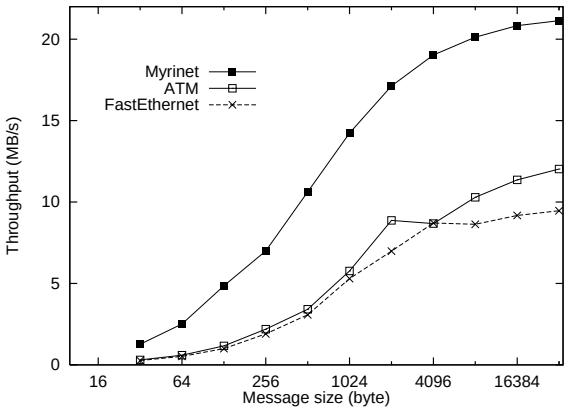


Figure 5. All-to-all throughput (on 8 nodes).

ing three communication kernels: ring, all-to-all exchange, and personalized all-to-all exchange.

The experiments reported below are done with 1,000 to 10,000 exchanges for each specified message size. For larger message sizes, it sometimes turned out to be profitable to synchronize the machines with a barrier between all exchanges. This limits pipelining in the network, but also limits contention for the network and switch channels.

In the 'ring' kernel, each node repeatedly exchanges messages with its two neighbors using Panda's message-passing primitives. The throughput results of the ring on 8 nodes are presented in Figure 4.

The performance on (non-switched) Fast Ethernet is poor. The bus-like network structure prevents independent communications to occur concurrently, hence, the total throughput of the 8 node ring is equal to the basic message passing throughput (10 MB/s). ATM and Myrinet are each interconnected by a switch, which allows for communication to occur in parallel. The ring throughput on ATM (50 MB/s) is about four times the individual message-passing throughput between a pair of nodes (13 MB/s). The best we can expect is a factor of four (four pairs communicating in parallel), so the ATM switch proves to be effective. A similar result is obtained on Myrinet. The aggregate ring throughput (127 MB/s) is three times that of the message-passing throughput (42 MB/s).

In the all-to-all kernel, each node sends a multicast message to all other nodes. This kernel in particular stresses the aggregate bandwidth provided by the underlying network. The numbers represent sender throughput, the receiver throughput is again 7 times higher.When comparing the all-to-all throughput presented in Figure 5 to the Panda group throughput from Figure 3, we find a strong similarity. The only significant difference is that on Myrinet the all-to-all kernel achieves higher throughput for large messages than the group communication: 21 MB/s versus 18 MB/s. The reason is that multicasts on both Fast Ethernet and

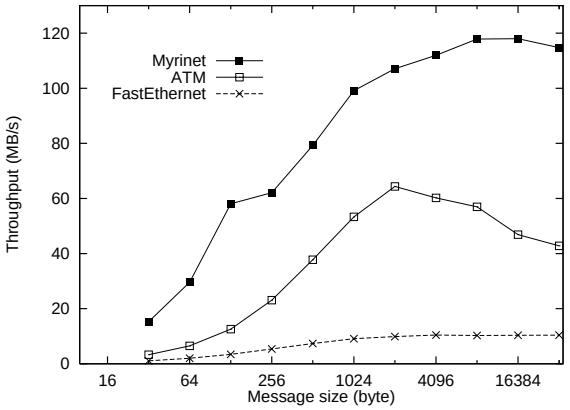


Figure 6. Personalized all-to-all throughput (on 8 nodes).

ATM have to share a single hardware resource (the hub and switch, respectively), while forwarding takes place in parallel in Myrinet's crossbar switch. We have measured the effect of our firmware optimizations. Switching off the forwarding in firmware reduces the maximum throughput with 30% to 14 MB/s. The knee in the ATM curve (at 4096 bytes) is caused by the fragmentation/reassembly code. The protocol on Fast Ethernet uses a packet size of 8196 bytes.

The third communication kernel performs a personalized all-to-all exchange where each node sends a message to all others. This type of communication behavior occurs frequently in numerical applications that need to transpose a matrix. The throughput on Fast Ethernet is limited to the 100 Mbit/s links speed (see Figure 6). A much higher throughput is achieved on ATM (64 MB/s) and Myrinet (118 MB/s). The decrease in performance on ATM for large message sizes is caused by the interconnection switch that occasionally drops a packet when it is queued for too long. Note the fragmentation/reassembly performance hit for Myrinet at the packet size of 256 bytes.

The performance plots of the three kernels show that Panda on Myrinet outperforms the others by far. This was to be expected given the low-level performance numbers. The relative difference, however, is increased since Myrinet takes full advantage of the parallelism offered by the crossbar switch. The ATM switch does provide parallel communication paths, but struggles with large messages. Fast Ethernet offers no parallel communication at all, which severely limits the throughput. The following sections will show how the differences affect application performance.

5. Challenging applications

With the applications listed in Table 1, we ran experiments to assess the performance benefits of fast networks. We selected the applications based on the criteria that they be challenging to the communication subsystem, i.e. they perform poorly on traditional local area networks. These communication-intensive applications typically achieve an efficiency of less than 25% on an 8-node workstation cluster with 10 Mb/s Ethernet. We have collected a wide variety of applications coded in different (parallel) languages (C, Fortran, Orca) using different message passing libraries (PVM, Panda, MPI). All Orca applications (Awari, Barnes-Hut, SOR, Monte Carlo, and Polygon Overlay, and Skyline Matrix) were developed at the Vrije Universiteit (see [2, 15, 19]). The Shallow Water application comes bundled with the ParkBench collection [8].The 3D-FFT code stems from a benchmark of PVM programs used to study the effectiveness of the TreadMarks system [6]. Finally, the Integer Sort and Multi-Grid applications were taken from the NAS Parallel Benchmarks [1].

Table 1 lists the important characteristics of our challenging applications: input set, sequential execution time for the given input set, communication pattern, and communication statistics. The total data volume and number of messages exchanged by each application were measured by instrumenting the Panda communication layer and measuring a test run on 8 nodes of our Pentium Pro cluster. The performance characteristics show that the benchmark applications stress the underlying communication subsystem indeed. Integer Sort, for example, requires an aggregate network throughput of at least 75.8 MB/s to be able to achieve linear speed-up on 8 nodes; this throughput number is computed by dividing the data volume by the ideal parallel execution time (154.4/(16.3/8) = 75.8 MB/s).

6. A simple performance model

To verify our measurements, we developed a simple scheme that estimates parallel performance in terms of sequential execution time and communication demands of a given application. The communication demands are derived from the performance data measured for the dominating low-level communication primitive or communication kernel. The scheme consists of the following steps:

1. We compute the average message size by dividing the total data volume by the number of messages (as listed in Table 1).
2. We use the average message size to look up the network throughput in the performance plot for the communication pattern of the application. The communication pattern is listed in Table 1, and the corresponding performance plot is presented either in Section 3 (low-level primitives) or Section 4 (communication kernels). Making the distinction between primitives and kernels is important since only the communication kernels take the network topology into account.
3. We use this throughput number to calculate a lower bound for the time spent communicating by dividing the data volume by the delivered throughput.
4. Finally, we use this communication time (comm_p) to estimate the parallel efficiency by assuming that the execution time on p processors equals the sequential execution time (T_1, listed in Table 1) divided by p plus the communication time:
$$\text{eff} = \frac{T_1}{p \cdot T_p} = \frac{T_1}{p \cdot (T_1/p + \text{comm}_p)} = \frac{T_1}{T_1 + p \cdot \text{comm}_p}$$

The above scheme provides a coarse performance model for applications on our workstation cluster using one of the fast networks. In particular, the model makes a number of assumptions that may not hold for realistics programs. First, the model assumes that the sequential code can be divided evenly among the parallel processes; in practice, most applications contain some sequential code that cannot be parallelized. Also, this assumption fails if the application suffers from load imbalance. Consequently, the model overestimates efficiency. Second, the model neglects data dependencies between communications, which may lead to additional waiting time in the application. This assumption also causes the model to overestimate application performance. Third, the model assumes that computation and communication occur separately, which may occur concurrently in some cases. For example, communication between one pair of nodes does not prevent the remaining nodes from computing. This last factor causes the model to underestimate application efficiency.

More refined performance models exist (e.g., LogP [7]), but our simple scheme already suffices to analyze the measurements that we performed on the Pentium Pro cluster.

7. Results

The measurements of the applications on the different networks and the corresponding performance estimates are

application	input	communication pattern	sequential time [s]	communication vol. [MB]	communication #msgs [10^3]
Integer Sort	class 6	(pers.) all-to-all	16.3	154.4	2.0
Shallow Water	small	hypercube+ring	535.0	2358.5	22.3
Multi-Grid	class S	ring	0.5	1.9	3.1
Monte Carlo	600	group comm.	52.0	116.0	77.9
SOR	480x1000	ring	63.9	64.8	17.3
Polygon Overlay	50Kx50K	RPC	8.6	9.1	0.4
3D-FFT	6,6,6,6	pers. all-to-all	22.2	24.5	0.5
Skyline Matrix	3000	RPC+group	36.9	33.0	21.9
Barnes-Hut	8192	group comm.	68.1	25.4	302.3
Awari	10	RPC	46.5	7.7	35.5

Table 1. Application characteristics: input set, sequential execution time, communication pattern, and message traffic measured on 8 nodes.

presented in Figure 7. It shows the efficiency on each of the three fast network configurations for a parallel run on 8 nodes. The sequential time was measured by running the parallel code on 1 processor; we did not use a special sequential version since this was frequently not available.

The performance bars in Figure 7 show that all but one application achieve an efficiency of over 50% on Myrinet. This is a good result, since the benchmark suite consists of challenging applications that perform badly (i.e. efficiency falls below 25%) on traditional 10 Mbit/s Ethernet. Given the basic performance measurements in the previous sections it is no surprise that the applications perform less well on ATM and Fast Ethernet. The applications are sorted on relative communication requirement from high (left) to low (right); the relative communication requirement is calculated by dividing the data volume by the sequential execution time of an application (see the data in Table 1). The gap between Myrinet, ATM, and Fast Ethernet is largest for the applications with the highest relative communication requirement (i.e. on the left in Figure 7), which matches the intuition. Barnes-Hut is the exception to this general observation. The poor performance on ATM and Fast Ethernet is caused by the exceptionally high number of small messages sent (see Table 1).

The performance model outlined in Section 6 yields relatively good estimates for the application efficiency on a given network. The model is very simple, yet most estimates are within 10% of the measured values. When an estimate differs significantly from the corresponding measurement, the estimate is usually too high because the model does not take into account application specifics like load imbalance. On the positive side, a too high estimate indicates that the application is not limited by the communication volume, so there is room for improvement. Note that most overestimates are for applications running on Myrinet, so most room for improvement is on Myrinet.

In the remainder of this section we discuss each application in some detail, and attempt to explain any discrepancies between the estimates and measurements. For applications using group communication, we report on the performance gain of our Myrinet firmware multicast implementation in comparison to a version using the original FM firmware.

7.1. Integer Sort

Integer Sort is part of the Nas Parallel Benchmark 2.2 (NPB2.2). It is written in C, with calls to MPI. We translated the MPI calls into equivalent PVM calls by hand. Integer Sort performs ranking of 2^{22} integers in the range $0 \ldots 2^{18}$. The goal is to calculate the (global) rank of each key. First, the keys are equally distributed over the processors, which each count the occurrence of their keys. This distribution is communicated to processor 0, which then calculates and broadcasts an equal distribution of key ranges over the processors. Then, each processor requests the rank of its keys from the processors that own the respective key ranges. All requests for each processor are collected into one message.

This algorithm involves RPC exchanges between all pairs of processors. The program is communication bound: on Myrinet 1.3 s is spent in communication and 2.0 s in computing. Consequently, the performance prediction with our approximation scheme is very close.

7.2. Shallow Water

The Shallow Water application (known as PSTSWM) is a message-passing parallel PVM Fortran code that solves the nonlinear shallow water equations on a rotating sphere using spectral transform methods. The nonlinear shallow water equations constitute a simplified atmospheric-like fluid prediction model that exhibits many of the features of more complete models, like climate models. The application consists of several consecutive transformations (Fast Fourier

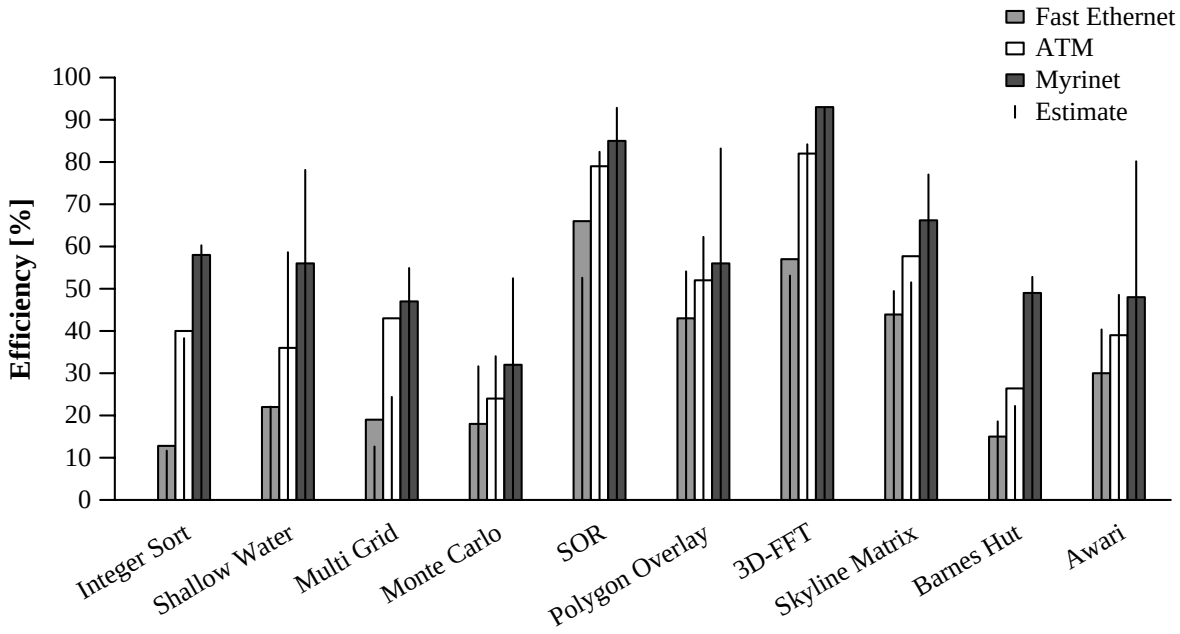


Figure 7. Application performance on 8 nodes: measurements (bars) and estimates (spikes).

and Legendre), which require hypercube and ring neighbor exchange of fairly large messages (over 100KB).

Shallow Water is the most demanding application of our benchmark suite: the total data volume exchanged over the network is 2.4 Gbyte. Since it also has considerable computational demands, the efficiency on Myrinet on 8 cpus is still 56%. For the other networks, the efficiency is much lower.

7.3. Multi-Grid

Multi-Grid was published as part of NPB2.1. It is a Fortran program which calls the MPI message passing library. Bundled with NPB2.1 comes a stubs library to translate Fortran MPI calls into Fortran PVM calls. This library was used to call the Panda PVM library.

This application solves a discrete Poisson equation using the V-cycle multigrid algorithm. Data points are partitioned among the processors in such a way that border points must be exchanged in one dimension after each iteration. Moreover, the program pushes residuals and pulls solutions from dense to coarser grids.

Multi-Grid is remarkable in that it runs for a very short time for the given input set, but is nevertheless so communication-intensive that it shows significant performance differences for the different networks. Fast Ethernet is uncapable of offering the required bandwidth for the underlying ring communication. On ATM and Myrinet Multi-Grid achieves an efficiency of just over 40%.

7.4. Monte Carlo

This Orca application is used to study an important problem in radiation chemistry: ion recombination in nonpolar liquids. The implementation uses a parallel Monte Carlo simulation for this computationally intensive task. Like N-body simulations, the application distributes the particles over the processors. It is harder to parallelize efficiently, however, since so-called *recombinations* (pairwise annihilations of particles) play an important role, causing the number of particles N to decrease during the simulation. To limit excessive communication for few particles, the number of workers is decreased and the particles are redistributed. This causes a sharp decrease of the potential parallelism during the application run, which accounts for the gap between measurements and estimates in Figure 7.

The Monte Carlo application uses group communication to propagate ion state changes (e.g, position, velocity) to all workers. Our firmware multicast on Myrinet has a clear impact on Performance. Monte Carlo runs 11% faster than a version implemented on top of the original FM software.

7.5. Successive OverRelaxation

Successive OverRelaxation (SOR) is a well-known finite-element method that iteratively updates all elements of a matrix. The application written in Orca assigns a horizontal strip of rows to each processor. At the end of each iteration, processors that own adjacent strips exchange boundary rows using two shared buffers (accessed by

RPCs). The application iterates until the difference with the previous matrix falls below a threshold.

Our model predicts larger performance differences for the networks than we actually measure. This can be attributed to the fact that, during the computation phases, communication and computation may be overlapped. The algorithm has synchronization phases which limit the overall parallelism; this explains why the estimated efficiency on Myrinet is higher than the measured efficiency.

7.6. Polygon Overlay

This application is one of the Cowichan problems [20]. It is derived form the spatial information systems domain: two maps covering the same geographical area must be overlaid to create the geometric intersection. A map is composed of a number of non-overlapping polygons, and is typically quite large: this application uses two maps containing 50,000 polygons each. The application is structured as a master that divides the maps into patches and distributes them to the worker processes. Each worker sorts its patches for efficiency and computes the polygons that constitute the intersection of the two patches, and sends the result back to the master. Since the maps are large, and the computation takes modest time, the performance is dominated by the communication subsystem. This application suffers a sequential bottleneck phase, since the list of polygons is created, distributed, and recombined by one processor. This explains the overestimated efficiency in Figure 7.

7.7. 3D-FFT

3D-FFT is a numerical program that solves a partial differential equation by applying forward and inverse FTTs. The computation is optimized by transposing the data in between the two phases. The matrix transpose, which is a personalized all-to-all exchange, has a noticeable impact on the overall performance of 3D-FFT running on Fast Ethernet (see Figure 7). This is to be expected since the measurements in Section 4 showed that the Fast Ethernet hub becomes a bottleneck for this communication pattern. The network congestion results in an efficiency of 57% for 3D-FFT on Fast Ethernet in comparison to 93% on Myrinet. Although the ATM switch also has been measured to be unable to cope with personalized all-to-all exchanges, the performance impact is less severe than for Fast Ethernet; 3D-FFT achieves an efficiency of 82% on ATM.

7.8. Skyline matrix solver

This Orca application solves a linear set of equations ($Mx = b$) for so-called skyline matrices that have a prefix of leading zeros in each row and column. Instead of using the standard elimination method for LU decomposition, this application uses a parallelized version of "Doolittle's method" that exploits the special matrix structure. The rows and columns are distributed cyclically to avoid load imbalance, and at each step the owning node broadcasts the pivot row and column used for elimination. Before and after the elimination phase RPCs are used to distribute the matrix and gather the results.

The Skyline program achieves acceptable performance on all three networks; even on Fast Ethernet the program achieves an efficiency of 44%. We have not been able yet to determine why the performance model underestimates the performance on ATM, while overestimating the performance on Fast Ethernet and Myrinet.

Since Skyline is using group communication, we performed an additional run on Myrinet to determine the performance gain of our multicast in firmware. Skyline runs 3% faster on the optimized firmware.

7.9. Barnes-Hut

This application is part of the SPLASH benchmark suite [16]. It solves an n-body problem; the main datastructure is an oct-tree that describes the location and velocity of each body. The original shared memory code has been rewritten to fit the distributed memory model provided by the Orca language. The bodies are partitioned over the processors. However, the oct-tree is represented as one shared Orca object, which has proven to be much more efficient than mapping each body to an individual Orca object. For each body, the forces are computed by traversal of the oct-tree to find all bodies that are close enough to contribute to the local force. During the computation phase the oct-tree object is therefore accessed very frequently to read information. Consequently, the Orca runtime system replicates the object. For each body, the newly calculated forces are updated on-the-fly, thus retaining the fine-grained character of this application. After the force calculation is completed for all objects, all processors calculate the new position and velocities for their bodies, and update them in the oct-tree.

For Barnes-Hut, the group throughput for small messages is most important. This application performs much better on Myrinet than on the other two networks (see Figure 7). This is caused by the advanced multicast support offered by our extensions to the Myrinet firmware. We have measured that our multicast implementation in firmware causes Barnes to execute 12% faster than on the original FM software for Myrinet.

7.10. Awari

This application generates an end-game database for the two-player boardgame Awari. The database is partitioned among the processors. During the construction of the end-

game board positions, information is propagated to the correct database entry. A hashing scheme is used to determine the location of each entry. This policy has the effect that most updates are propagated to a remote part of the database. To avoid swamping the network with many small messages, each holding a single update, the Orca program aggregates multiple updates to the same database into one message. This optimization is mandatory for efficiency, see [2]; however, it also throttles the parallelism.

Awari exhibits a chaotic communication pattern: the Orca runtime system issues RPCs to arbitrary destinations to propagate the database updates. The difference in estimated and measured efficiency can be attributed to a number of causes. First, to the message combining throttle; second, to a complicated distributed termination algorithm which is absent in the sequential code, and last, to the fact that messages often arrive at a busy processor, thus causing an interrupt. Parallelism is limited because the end-game database that is computed is rather small (10 stones); when computing larger end-game databases parallel efficiency increases (see [2]). Nevertheless, a far higher efficiency is achieved on Myrinet than on Fast Ethernet or ATM.

8. Discussion and related work

The application measurements show that good performance can be achieved on Myrinet. Performance on ATM is somewhat lower, but not as low as might be expected based on the difference in kernel communication performance. The performance gap between Myrinet and Fast Ethernet is considerable for most applications. In particular applications that benefit from parallel communications (like a ring or personalized all-to-all exchange) are hampered by the fact that Fast Ethernet is interconnected by a simple hub. Since the low-level latencies and throughputs achieved by FastEthernet are similar to those of ATM, we expect that upgrading the Fast Ethernet network with a switch will increase application performance to approximately the same level. For our Pentium Pro cluster, however, including a switch was not considered since Fast Ethernet is normally used for file server traffic only.

The simple performance model based on an application's communication pattern, message size, and corresponding throughput number provides a reasonable estimate of the measured performance on the network at hand. This is a consequence of most applications being bounded by the underlying throughput, not the latency of the networks. We feel that the simple model could be used to provide performance feedback to the application programmer. If the estimated efficiency differs from the measured performance, the model indicates that the application performance is not limited by communication, but by something else, so there is room for improvement. Additional experimental evaluation of application performance is necessary to determine if the model is also useful as a performance indicator for other classes of applications and for other parallel machines with fast interconnects. Some important machine parameters are communication-to-computation ratio, number of nodes, and bi-section bandwidth.

An interesting experiment by Martin et al. studies the impact of latency and bandwidth on applications by modifying the Myrinet firmware to insert delays and reduce bitrate [11]. Their main conclusion from a set of controlled experiments is that applications are most sensitive to latency, in particular to time spent in communication protocols on the host, and surprisingly tolerant to bandwidth. Our experiments, however, indicate that the challenging applications are bandwidth limited. The simple model based on data volume and communication pattern provides reasonable estimates of performance, hence, required bandwidth is the key performance characteristic. A possible explanation for these conflicting observations is that the two sets of applications were written in different programming models. The applications used by Martin et al. are written in Split-C, a language that provides a global address space on distributed memory machines. In contrast, our benchmark programs are mostly written as coarse-grained message passing programs. Recent work by Chong et al. on the Alewife machine shows that message passing programs are relatively insensitive to latency in comparison to shared-memory programs [5]. This is a consequence of message passing programs exploiting asynchronous communications to tolerate network latency by overlapping communication and computation.

9. Conclusions

We investigated the feasibility of clusters of workstations to run challenging parallel applications. Since the arrival of fast interconnection networks, such clusters seem competitive with commercial MPPs, although at a fraction of the cost. We equipped an 8-node Pentium Pro cluster with two fast networks: 155 Mbit/s ATM and 1.28 Gbit/s Myrinet. For reference, we included measurements on a modern low-cost LAN: 100 Mbit/s Fast Ethernet. Fast Ethernet is accessed through UDP/IP implemented by the BSD/OS kernel. ATM and Myrinet are directly accessed from user-space. Although this difference blurs the raw network performance, it is the combination of software and hardware that is most relevant to application programmers.

Of the three networks investigated, Myrinet provides by far the best base performance, followed at a distance by ATM, again followed by Fast Ethernet. The high speed of the Myrinet links is not the only reason for success; the programmability of their network interface allows for high-performance software such as Fast Messages to take full advantage of the hardware capacities. For the performance

on communication kernels (ring, all-to-all, personalized all-to-all), the gap between Fast Ethernet and ATM/Myrinet widens, since the Fast Ethernet hub does not support communications to occur in parallel as do the ATM switch and Myrinet crossbar.

Since application performance is what really matters, we investigated 10 challenging parallel applications with considerable communication demands. For all applications, the parallel performance showed a marked difference on the different networks, Myrinet each time emerging the clear winner. Our Myrinet firmware extensions contributed to this success; we measured that the multicast in firmware boosts performance up to 12%. To aid in discussing individual application results, we developed an extremely simple model that (over)estimates the performance of parallel applications, given their computation and communication demands. We used this model to indicate outlyers that needed further discussion.

Based on the application measurements we conclude that Myrinet does shift the border of which applications can be run efficiently on clusters of workstations. On the other hand, the model's estimates indicate that some applications can benefit from still faster networks.

Acknowledgements

We thank Aske Plaat, Tim Rühl, Kees Verstoep, and the anonymous referees for commenting on draft versions of this paper. Kees also developed most of the low-level software accessing the network device drivers. We thank Raoul Bhoedjang for porting and fine-tuning the 3D-FFT program. Finally, we thank Andrew Chien and Scott Pakin for making the FM software available to us.

References

[1] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. Technical Report NAS-95-020, NASA Ames Research Center, Dec 1995.

[2] H. Bal and L. Allis. Parallel Retrograde Analysis on a Distributed System. *Supercomputing '95*, Dec. 1995.

[3] H. Bal, M. Kaashoek, and A. Tanenbaum. Orca: A Language for Parallel Programming of Distributed Systems. *IEEE Transactions on Software Engineering*, 18(3):190–205, Mar. 1992.

[4] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[5] F. Chong, R. Barua, F. Dahlgren, J. Kubiatowicz, and A. Agarwal. The Sensitivity of Communication Mechanisms to Bandwidth and Latency. In *HPCA-4*, Las Vegas, Nevada, Feb. 1998.

[6] A. Cox, S. Dwarkadas, H. Lu, and W. Zwaenepoel. Message Passing Versus Distributed Shared Memory on Networks of Workstations. *Supercomputing '95*, Dec. 1995.

[7] D. Culler, R. Karp, D. Patterson, A. Sahay, E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken. LogP: A Practical Model of Parallel Computation. *Communications of the ACM*, 39(11):78–85, Nov. 1996.

[8] J. J. Dongarra and T. Hey. The ParkBench benchmark collection. *Supercomputer*, 11(2-3):94–114, June 1995.

[9] M. Kaashoek. *Group Communication in Distributed Computer Systems*. PhD thesis, Vrije Universiteit, Amsterdam, Dec. 1992.

[10] K. Langendoen, J. Romein, R. Bhoedjang, and H. Bal. Integrating Polling, Interrupts, and Thread Management. In *Frontiers'96*, pages 13–22, Annapolis, Maryland, Oct. 1996.

[11] R. Martin, A. Vahdat, D. Culler, and T. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *The 24th Annual International Symposium on Computer Architecture*, Denver, Colorado, June 1997.

[12] S. Pakin, V. Karamcheti, and A. Chien. Fast Messages: Efficient, Portable Communication for Workstation Clusters and MPPs. *IEEE Concurrency*, 5(2):60–73, Apr. 1997.

[13] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Supercomputing '95*, San Diego, CA, Dec. 1995.

[14] T. Rühl, H. Bal, R. Bhoedjang, K. Langendoen, and G. Benson. Experience with a Portability Layer for Implementing Parallel Programming Systems. In *The 1996 Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'96)*, pages 1477–1488, Sunnyvale, California, August 1996.

[15] F. Seinstra, H. Bal, and H. Spoelder. Parallel Simulation of Ion Recombination in Nonpolar Liquids. In *High Performance Computing and Networking (HPCN'97)*, Vienna, Austria, Apr. 1997.

[16] J. Singh, W.-D. Weber, and A. Gupta. SPLASH: Stanford Parallel Applications for Shared Memory. *ACM Computer Architecture News*, 20(1):5–44, Mar. 1992.

[17] K. Verstoep, K. Langendoen, and H. Bal. Efficient Reliable Multicast on Myrinet. In *Proc. 1996 Int. Conf. Parallel Processing (Vol. III)*, pages 156–165, Bloomingdale, Ill., Aug. 1996.

[18] R. Vetter. ATM Concepts, Architectures, and Protocols. *Communications of the ACM*, 38(2):30–38, Feb. 1995.

[19] G. Wilson and H. Bal. An Empirical Assessment of the Usability of Orca Using the Cowichan Problems. *IEEE Parallel and Distributed Technology*, 4(3), Fall 1996.

[20] G. V. Wilson. Assessing the Usability of Parallel Programming Systems: The Cowichan Problems. In *Proceedings of the IFIP Working Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 183–193. Birkhäuser Verlag AG, April 1994.