In: Proceedings of the International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'96), August 9-11th 1996, Sunnyvale, CA, Vol. I, pp. 375-386, best paper award.

The ParaStation$^{(TM)}$ Project: Using Workstations as Building Blocks for Parallel Computing

Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy
University of Karlsruhe, Dept. of Informatics
Am Fasanengarten 5, 76128 Karlsruhe, Germany
email: {warschko,blum,tichy}@ira.uka.de
URL: http://wwwipd.ira.uka.de/parastation

Abstract

The ParaStation communication fabric provides a high-speed communication network with user-level access to enable efficient parallel computing on workstation clusters. The architecture, implemented on off-the-shelf workstations coupled by the ParaStation communication hardware, removes the kernel and common network protocols from the communication path while still providing full protection in a multiuser, multiprogramming environment.
The programming interface presented by ParaStation consists of a UNIX socket emulation and widely used parallel programming environments like PVM, P4, and MPI. This allows porting a wide range of client/server and parallel applications to the ParaStation architecture. The first implementation of ParaStation using Digital's AlphaGeneration workstations achieves a communication latency as low as $2.5\mu s$ (process-to-process) and a sustained bandwidth of more than 10 Mbyte/s per process. Benchmarks using PVM on ParaStation demonstrate real application performance of 1 GFLOP on an 8-node cluster.

Keywords: Workstation Cluster, Parallel and Distributed Computing, User-Level Communication, High-Speed Interconnects.

1 Introduction and Motivation

Workstation clusters in conjunction with high-speed interconnection networks offer a cost-effective and scalable alternative to monolithic supercomputers. In contrast to supercomputers and parallel machines, clustered workstations rely on standardized communication hardware and communication protocols developed for local-area networks and not for parallel computing. As communication hardware is getting faster and faster, the communication performance is now limited by the processing overhead of the operating system and the protocol stack, rather than the network itself. Thus, users who upgrade from ethernet to, e.g., ATM failed to observe an application speedup comparable to the improvement of the raw communication performance. The peak bandwidth in high-speed networks is often achieved only with extremely large data streams, and there is no improvement at all when communication is based on relatively small data units. Exchanging relatively small messages, however, is a key issue in parallel computing. Besides, this holds also for the object-oriented cooperative approach on platforms like CORBA. Objects as communication units result in transmitting rather small messages between cooperating nodes. Another network-related problem within parallel computing is the scalability of the network when adding more and more stations. A typical SPMD-like parallel program consists of a sequence of computing and communicating parts. As a consequence, all nodes involved either compute intermediate results, using no network at all, or they exchange data with each other, generating a burst-traffic situation on the network. Burst traffic in traditional LAN topologies sharing one physical medium (e.g., bus and ring) results in a worst-cast behaviour of the network which means large latencies and less throughput. In contrast, MPP networks use higher-dimensional topologies such as grids or hypercubes, where bandwidth and bisection-bandwidth increases with the number of connected nodes. Thus, critical parameters for a network well-suited for parallel computation are

- latency, when transmitting small messages,
- throughput, when transmitting large data streams, and
- scalability, when increasing the number of connected workstations.

In addition to these hardware-related parameters, the functional behavior of the network should allow for implementing reduced or even minimal protocol stacks. To avoid processing overhead in a protocol stack, the underlying network has to maintain as much functionality as possible. In contrast to most local-area networks, MPP networks offer reliable data transmission in combination with hardware-based flow control. As a consequence, traditional protocol functionality such as window-based flow control, acknowledgement of packages, and checksum processing can be reduced to a minimum (or even be left out). If the network further guarantees in-order delivery of packets, the fragmentation task and especially the defragmentation task of a protocol is much easier, because incoming packets don't have to be rearranged into the correct sequence. Implementing as much protocol-related functionality as possible directly in hardware results in minimal and thus efficient protocols.

The most promising technique to improve the performance of the network interface as seen by a user is to move the protocol processing into the user's address space. The ParaStation system in fact does all protocol processing at user level while providing full protection. Another critical issue is the design of the user interface to the network. Often vendors support proprietary APIs (e.g., AAL5 for ATM), but for reasons of portability a user would prefer a standardized and well-known interface. Thus, the key issues for the design of an efficient communication subsystem for the ParaStation architecture are

- sharing the physical network among several processes,

- providing protection between processes using the network simultaneously,
- removing kernel overhead and traditional network protocols from the communication path, and
- providing a well-known programming interface.

To reach the goal of efficiency, the kernel is removed from the communication path and all hardware interfacing and protocol processing is done at user level. Even protection is done within the system library at user level using processor-supported atomic operations to implement semaphores. Besides proprietary user interfaces, we decided to provide an emulation of the standard Unix socket interface on top of our system layer.

This paper describes the ParaStation architecture and related approaches (section 2), starting with a description of the ParaStation network (section 3) and the ParaStation software architecture (section 4). Various benchmarks are presented in section 5.

2 Related Work

There are several approaches targeting efficient parallel computing on workstation clusters which can be classified as shared-memory and distributed-memory systems. Shared-memory systems such as MINI [11], SHRIMP [4], SCI-based SALMON [10, 13], Digital's MemoryChannel [15], and Sun's S-Connect [12] support memory-mapped communication, allowing user processes to communicate without expensive buffer management and without system calls across the protection boundary separating user processes from the operation system kernel.

In contrast to these approaches, distributed memory systems such as Active Messages [16] for ATM-based U-Net [2], Illinois Fast Messages [14] for Myrinet [5], and the Berkeley NOW project [1] focus on a pure message-passing environment rather than a virtual shared memory. As von Eicken et al. pointed out [16], recent workstation operating systems do not support a uniform address space, so virtual shared memory is difficult to maintain.

As with Active Messages and Fast Messages, performance improvement within the ParaStation system is based on user-level access to the network, but in contrast to them, we provide multiuser/multiprogramming capabilities. Like Myrinet and S-Connect, our network was originally designed for a MPP System (Triton/1) and is now retargeted to a workstation cluster environment. Myrinet, IBM-SP2, and Digital's Memory Channel use central switching fabrics, while ParaStation provides distributed switches on each interface board.

3 The ParaStation Network

The network topology is based on an two-dimensional toroidal mesh. For small systems, a ring topology is sufficient. Data transport is done via a table-based, self-routing packet switching method which uses virtual cut-through routing. Every node is equipped with its own routing table and three input buffers (see figure 1): two for intermediate storage of data packets coming from other nodes and one for receiving packets from its associated processing element (workstation). An output buffer delivers data packets to the associated workstation. The buffering decouples network operation from local processing. Packets contain the address of the target node, the number of data words contained in the packet, and the data itself. The size of the packet can vary in the range from 4 to 508 bytes. Packets are delivered in order and no packets will be lost. Flow control is done at link level and the unit of flow control is one packet.

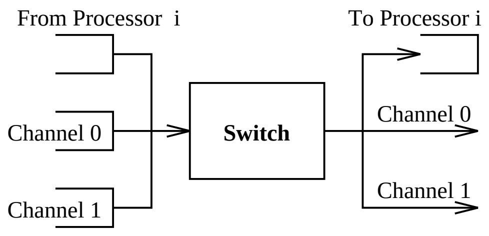

Figure 1: ParaStation network

For both topologies – ring and toroidal mesh – we provide a deadlock-free routing scheme. Deadlock-free routing on a ring is simple as long as the network is prevented from overloading. Inserting new packets into the network only when both channel fifos are empty solves this problem. Deadlock-free routing on a toroidal mesh is done by using X-Y dimension routing. A packet is routed along the x axis of the grid first until it reaches its destination column. Then it is routed along the y axis to its final destination node. Providing similar insertion rules as in the ring routing scheme for both dimensions and giving the y axis priority over the x axis prevents deadlocks.

The current implementation of our communications processor involves a routing delay of about $250ns$ per node and offers a maximum throughput of 20 Mbyte/s per link. Additionally, the interface board provides a hardware mechanism for fast barrier synchronization. To connect several systems, we use 60-pin flat cables, with standardized RS-422 differential signals. Using this technology, the maximum distance between two systems is $10m$.

4 The ParaStation Architecture

The goal is to support a standardized, but efficient programming interface like UNIX sockets on top of the ParaStation network. The ParaStation network is dedicated to parallel applications and is not intended as a replacement for a common LAN, so associated protocols (e.g., TCP/IP) can be eliminated. These properties allow using specialized network features, optimized point-to-point protocols, and controlling the network at user level without operating system interaction (see figure 2). The ParaStation protocol software implements multiple logical communication channels on a physical link. This is essential to set up a multiuser/multiprogramming environment. Protocol optimization is done by minimizing protocol headers and eliminating buffering whenever possible. Sending a message is implemented as zero-copy protocol which transfers the data directly from user-space to the network interface. Zero-copy behaviour during message reception is achieved when the pending message is addressed to the receiving process; otherwise the message is copied once into a buffer in a common message area. Within

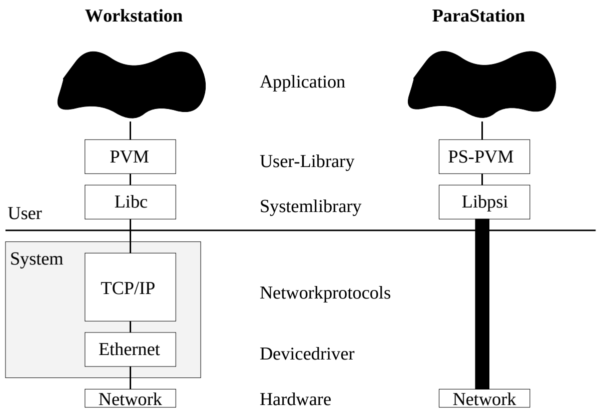

Figure 2: Network interfacing techniques

the ParaStation network protocol, operating system interaction is completely eliminated, removing it from the critical path of data transmission. The functionality missing to support a multiuser environment is realized at user level in the ParaStation system library.

The ParaStation system library (see figure 3) consists of three building blocks: the hardware interface layer, the central system layer, and the standardized user interface (sockets).

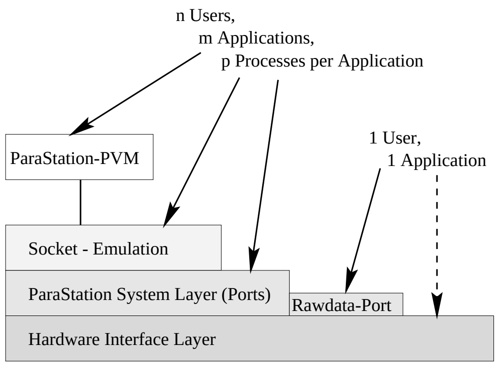

Figure 3: ParaStation system library

4.1 Hardware Layer

The hardware layer provides the necessary abstraction of the underlaying hardware to maintain a transparent and system independent interface to the upper layers. The functionality implemented in this layer consists of highly optimized send/receive operations, status information calls, and an initialization call. Information calls look for pending messages and check if the network is ready to accept new messages and initialization is used to map communication buffers into user space.

Since messages at this level are addressed to nodes rather than individual communication channels, message headers simply contain the address of the target node, the number of data words contained in the packet, and the data itself. When sending a message, data is copied directly from user-space memory to the interface board and the receiving function does the same vice versa, eliminating all intermediate buffering.

As a consequence, multiple applications using this layer are not supported, but it is nevertheless possible to use this layer as application programming interface.

4.2 System Layer (Ports)

The system layer establishes multiple communication channels between applications and supports a multiuser/multiprogramming environment. Instead of using operating system capabilities to set up a multiapplication interface, we decided to reassemble operating system functionality at user level to meet our primary design goal of efficiency.

To support individual communication channels (called *ports* in ParaStation), the system layer maintains a minimal software protocol which adds information about the sending and receiving port in each packet. This concept is sufficient to support multiple processes by using different port ids for different processes. Since message reception is done in user-space, and at least the protocol information has to be received to determine the destination port of the message, it is possible that process A receives a message addressed to a port which is owned by process B. To solve this problem, we use a common message area to buffer this kind of messages. To maintain a correct interaction between processes while sending or receiving messages, critical sections in this protocol layer are locked by semaphores. For reasons of efficiency, we also implemented these semaphores at user-level, using processor-supported atomic operations. Deadlock-free communication while sending large messages which cannot be buffered by the hardware is ensured by a combination of sending and receiving message fragments. Prerequisite for sending a message fragment is that the network will accept it. Otherwise incoming messages are processed first to prevent the network from overloading and blocking.

For performance reasons, a so-called *rawdata port* can be used to obtain as little overhead as possible. The *rawdata port* and upper layers can be used simultaneously, while *rawdata* applications are scheduled one after another.

As a result, the implementation of these concepts does not need system calls at all. Furthermore, we provide a zero-copy behavior (no buffering) whenever possible. This leads to high bandwidth and low latencies.

4.3 Socket Layer

The socket layer provides an emulation of the standard UNIX socket interface (TCP and UDP connections), so applications using socket communication can be ported to the ParaStation system with little effort. For connections outside a ParaStation cluster, regular operating system calls are used. `Send/recv` calls, which can be satisfied within the ParaStation-cluster, do not need any interaction with the operating system.

4.4 Application Layer

ParaStation implementations of standard programming environments like PVM [3], P4 [6], TCGMSG [9], or MPI (`mpich`) [8] use ParaStation sockets for high-speed communication. This approach allows us to easily port, maintain, and update these packages. We use the standard workstation software.

5 Benchmarks

The benchmarks described in this section cover three different scenarios. The communication benchmark provides information about the raw performance of ParaStation. Although we call this raw performance, the benchmark reflects application-to-application performance measured at the hardware interface layer. Second, we present the level of performance that can be achieved at ParaStation's different software layers. The third scenario deals with application performance, namely run-time efficiency. Low-level benchmarks were taken on two different DEC Alpha clusters (275-MHz 21064A processors and 233-MHz 21066 processors) as well as on a two-node Pentium system (120-MHz) running Linux (Version 1.3.94). For the application benchmarks we used an 8-node ParaStation cluster with 275-MHz DEC Alpha machines running Digital Unix 3.2c (OSF/1).

5.1 Communication Benchmark

To measure the end-to-end delay, we implemented a *Pairwise Exchange* benchmark, where two processes send a message to each other simultaneously, and then receive simultaneously. Unlike a *Ping-Pong* benchmark, process two does not wait for receipt of a message before transmitting. This is a more practical scenario for two processes exchanging messages. Benchmark results for three different systems are presented in figure 4.

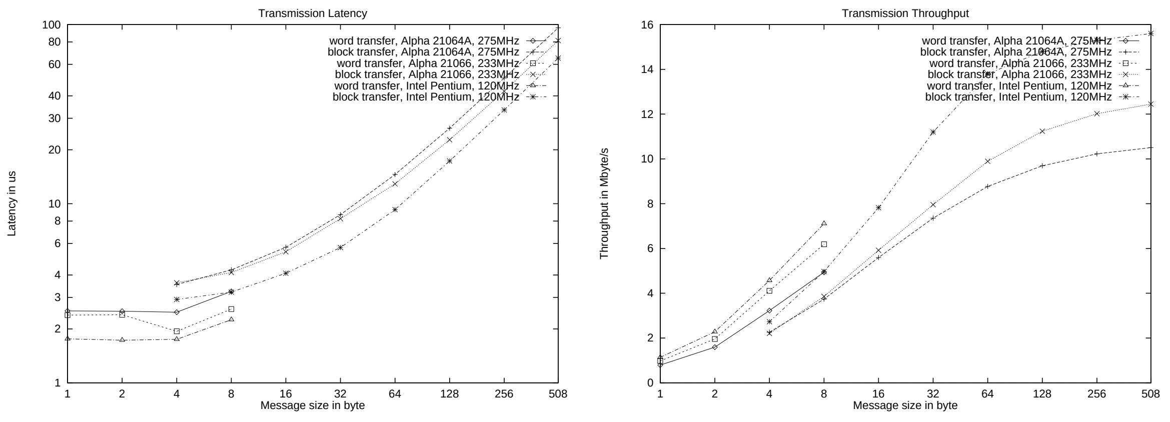

Figure 4: ParaStation communication benchmark

For small message sizes (4 byte), ParaStation achieves transmission latencies (sending and receiving a message in user space) as low as $2.5\mu s$ on systems with the 21064A, $1.9\mu s$ on systems with the 21066, and $1.7\mu s$ on systems with the Pentium processor. Thus, the latency for one communication operation – either a send or a receive – is just half of the presented numbers: $1.24\mu s$ for the 21064A, $0.97\mu s$ for the 21066, and $0.87\mu s$ for the Pentium system. For larger message sizes, when overhead per byte decreases, we get a total throughput of up to 10.5 Mbytes/s (21064A), 12.5 Mbytes/s (21066), and 15.6 Mbytes/s (Pentium) respectively. The performance differences are due to the internal architectures of the processors and the motherboard. The Alpha 21066 has the PCI interface on chip, whereas the Alpha 21064A is using a board-level chip set. In contrast to what would be expected looking at pure processor speed, the PC-based system outperforms the Alphas, although they are using a board-level chip set[1] as the 21064A systems does.

5.2 Performance of the Protocol Hierarchy

Switching from singleprogramming to multiprogramming environments often suffers from a drastic performance decrease. Table 1 presents performance figures of all software layers in the ParaStation system.

Table 1: ParaStation performance of various layers

	Alpha 21064A, 275 MHz				**Pentium, 120 MHz**			
	ParaStation		**OS/Ethernet**		**ParaStation**		**OS/Ethernet**	
protocol-layer	latency [μs]	band-width [MB/s]	latency [μs]	band-width [MB/s]	latency [μs]	band-width [MB/s]	latency [μs]	band-width [MB/s]
hardware	1.24	10.5			0.87	15.6		
rawdata	4.15	9.6			3.05	14.5		
port	8.85	8.9			7.65	11.7		
socket	11	8.8	283	0.99	7.65	11.7	159	1.08
P4	108	7.5	344	0.95				
PVM	129	6.7	539	0.84	102	7.7	388	0.86
socket (self)	6.4	85	195	33	4.82	88	288	30

To support a true multiprogramming environment, our system layer (ports) only adds about 7.6μs additional latency to communication calls, and the loss of throughput compared to the hardware abstraction layer on the Alpha system is within 15% (25% on the PC system). Furthermore, our decision to maintain the rawdata port is justified by the results shown. The rawdata port is twice as fast in latency than regular ports and the loss of throughput drops to 8.5% (7.5% on the PC system) compared to the performance of the hardware abstraction layer. 4.15μs (3.05μs) latency of the rawdata port is even less than 4.5μs (3.9μs) for a null system call on the Alpha (PC). Most of this time is used to guarantee mutual exclusion and correct interaction between competitive processes.

The real advantage of ParaStation becomes obvious when comparing the performance to that of regular operating system calls. ParaStation socket calls on the DEC Alpha are about 26 times faster in latency than the regular OS calls, while offering the same services. Similar results are measured on the PC system where ParaStation is about 21 times faster in latency than equivalent operating system calls. Throughput, however, is not comparable because the ParaStation network is much faster than Ethernet. Even the relative loss in throughput is not comparable because it is much harder to interface to a fast network than to a slower one.

Another interesting result is additional overhead caused by the programming environments P4 and PVM. Within ParaStation on the Alpha system, these environments add an overhead of factor 9.8 (P4) and 11.7 (PVM) to the latency of our system layer. Even in the standard operating system environment, P4 adds about 21% and PVM

[1]We use ASUS boards with an Intel Triton chip set.

116% overhead. Similar results are measured on the PCs where PVM adds an overhead of factor 13.3 to the latency and 144% overhead to the regular operating system, respectively. PVM even decreases throughput when built on top of the ParaStation sockets by 24% on the Alpha system and 34% on the PC system. This shows that both packages are not well designed for high-speed networks.

Finally, we measured the performance of a socket-to-socket communication within a single process, where no network hardware is needed at all. This test aims to measure the protocol performance for local communication in absence of process switching. Local communication on ParaStation is optimized and enqueues the send message directly into the receive queue of the receiving socket. Thus, the presented 85 Mbyte/s (88 Mbyte/s on the PCs) reflects mainly the `memcopy` performance of the system. The TCP/IP implementation within both Digital Unix and Linux seem to optimize local communication because a throughput of 33 MBytes/s (30 MBytes/s on the PC) is achieved with this benchmark test.

5.3 Application Benchmarks

Focusing on latency and throughput only is too narrow for a complete evaluation. It is necessary to show that a low-latency, high-throughput communication subsystem also achieves a reasonable application efficiency. Our approach is twofold. First, we took a *heat diffusion* benchmark to test application performance on our proprietary interface. Second, we installed the widely used and publicly available ScaLAPACK math library, which first uses the BLACS package and then PVM as communication subsystem. Thus, the complete protocol hierarchy as presented in the previous section is involved.

All application benchmarks were executed on an 8-node Alpha 21064A (275 MHz) cluster.

5.3.1 Heat Diffusion

The *heat diffusion* benchmark starts with an even temperature distributions on a metal plate. On all four sides heat sources and heat sinks are applied. The goal is to compute the final heat distribution of the metal plate. This can easily be done with a Jacobi or Gauss-Seidel iteration by calculating the new temperature of each gridpoint as average of its four neighbours.

Parallelizing this algorithm is simple: we use a block distribution of rows of the $n \times n$ matrix, so during each iteration each process has to exchange two rows with its neighboring processes. To visualize the progress, all data is periodically collected by one process. The following table shows the effective speedup for different problem sizes. Each experiment was measured with at least 5000 iterations, visualizing the progress every 20 iterations.

As shown in table 2, we achieve a reasonable speedup for relevant problem sizes on all configurations. Taking the last row as an example, the efficiency of two workstations is close to its maximum (97.5%). In the four and eight-processor configurations, we achieve speedups of 3.75 with 4 nodes and 7.4 with 8 nodes and therefore an efficiency of 93.75% (4 nodes) and 92.5% (8 nodes), respectively. The gap is caused by the visualization process which is an inherent sequential task. In general, there are only two points

Table 2: Heat diffusion benchmark

Problem size (n)	1 workstation Runtime [ms/iter]	2 workstations Runtime [ms/iter]	2 workstations Speedup	4 workstations Runtime [ms/iter]	4 workstations Speedup	8 workstations Runtime [ms/iter]	8 workstations Speedup
64	1.5	0.99	1.51	0.9	1.66	2.0	0.75
128	6.0	3.5	1.71	2.3	2.61	3.4	1.77
256	22.3	12.0	1.86	7.5	2.97	7.0	3.19
512	89.2	46.7	1.91	26.4	3.38	17.2	5.19
1024	424	217	1.95	113	3.75	57.3	7.40

where performance decreases when switching to the next larger configuration. But this only happens for small problem sizes where parallelizing is doubtful.

5.3.2 ScaLAPACK

The second application benchmark for ParaStation, *xslu* taken from ScaLAPACK, is an equation solver for dense systems. Numerical applications are usually built on top of standardized libraries, so using this library as benchmark is straightforward. Major goals within the development of ScaLAPACK [7] were efficiency (to run as fast as possible), scalability (as the problem size and number of processors grow), reliability (including error bounds), portability (across all important parallel machines), flexibility (so users can construct new routines from well-designed parts), and ease of use. ScaLAPACK is available for several platforms, so presented results are directly comparable to other systems.

Table 3: ScaLAPACK benchmark

Problem size (n)	1 workstation time [s]	1 workstation MFlop	2 workstations time [s]	2 workstations MFlop	4 workstations time [s]	4 workstations MFlop	8 workstations time [s]	8 workstations MFlop
1000	5.0	134	3.36	199	2.95	226	2.74	244
2000	34.4	155	20.8	257	13.6	394	9.80	545
3000	109	165	62.3	289	39.2	459	27.9	647
4000			138	309	84.0	508	54.6	782
5000					152	547	96.4	865
6000					251	573	157	920
7000							234	978
8000							334	**1022**
Ethernet	n=3000	165	n=4000	232	n=6000	320	n=8000	261

Table 3 confirms that performance scales well with problem size as well as number of processors. Comparing the achieved MFLOPs of the two, four, and eight-processor clusters to the maximum performance of a single processor (165 MFLOP) results in efficiency factors of 94% (2 nodes), 87% (4 nodes), and 77% (8 nodes), respectively. It is remarkable that we get more than a GFLOP for the 8-processor cluster. These are real measured performance figures and not theoretically calculated numbers. The last row shows the performance one can get using ScaLAPACK configured with standard PVM (Ethernet). The best performance in this scenario is reached at problem size of n=6000 on a 4-processor cluster. Using more processors results in a drastic performance loss due to bandwidth limitation on the Ethernet. For ParaStation, we see no limitation when scaling to larger configurations. And it is even possible to improve the ParaStation performance by using a better interface than PVM.

In general, using various application codes such as digital image processing and finite element packages, we achieved relative speedups of 3 to 5 on ParaStation over regular PVM or P4 on our 4-node and 8-node ParaStation clusters. In all of these studies, we used the same object codes, just linking them with different libraries.

6 Conclusion and Future Work

The integrated and performance-oriented approach of designing fast interconnection hardware and a system library with a well-defined and well-known user interface has lead to a workstation cluster environment that is well-suited for parallel processing. With low communication latencies, minimal protocol, and no operating system overhead, it is possible to build effective parallel systems using off-the-shelf workstations. While ParaStation is still a workstation cluster rather than a parallel system, the presented performance compares well to parallel systems. ParaStation's flexibility, scalability (from 2 to 100+ nodes), portability of applications (providing standard environments like PVM, P4, and Unix sockets), and the achieved performance level have led us to market ParaStation[2]. The additional cost per workstation for a ParaStation communication adapter is marginal[3] compared to the price of a fully equipped workstation. In contrast to other high-speed networks such as ATM and FiberChannel, there are no additional costs for central switching components within ParaStation.

In future, we will work on next-generation hardware, ports to other platforms, and support for various programming environments. Current issues for a new network design are fiber-optic links and flexible DMA engines to reach an application-to-application bandwidth of about 100 Mbyte/s. Second, due to the PCI-bus interface, the ParaStation system is not limited to Alpha platforms. Currently, we are working on the PC/Linux platform. PCs running Windows NT are scheduled and Alphas running either Linux or NT will follow. Finally, we plan to support optimized versions of MPI as a future standard as well as an optimized PVM as a de facto standard directly within the ParaStation system layer. This will give PVM/MPI applications a performance boost over a socket-based implementation. Besides PVM/MPI, Active Messages and Fast Messages are both considered as additional interfaces to the system layer.

References

[1] Thomas E. Anderson, David E. Culler, and David A. Patterson. A Case for NOW (Network of Workstations). *IEEE Micro*, 15(1):54–64, February 1995.

[2] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-net: A user-level network interface for parallel and distributed computing. In *Proc. of the 15th ACM Symposium on Operating Systems Principles, Copper Mountain, Colorado*, December 3-6, 1995.

[2]see: http://wwwipd.ira.uka.de/parastation

[3]2800 DM for the communication board, including all software.

[3] A. Beguelin, J. Dongarra, Al Geist, W. Jiang, R. Manchek, and V. Sunderam. *PVM 3 User's Guide and Reference Manual.* ORNL/TM-12187, Oak Ridge National Laboratory, May 1993.

[4] Matthias A. Blumrich, Cezary Dubnicki, Edward W. Felten, Kai Li, and Malena R. Mesarina. Virtual-Memory-Mapped Network Interfaces. *IEEE Micro*, 15(1):21–28, February 1995.

[5] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jarov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, February 1995.

[6] Ralph Buttler and Ewing Lusk. *User's Guide to the p4 Parallel Programmimg System.* ANL-92/17, Argonne National Laboratory, October 1992.

[7] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers – Design Issues and Performance. Technical Report UT CS-95-283, LAPACK Working Note #95, University of Tennesee, 1995.

[8] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI.* MIT-Press, 1994.

[9] R. J. Harrison. Portable tools and applications for parallel computers. *International Journal on Quantum Chem.*, 40:847–863, 1991.

[10] IEEE. *IEEE – P1596 Draft Document. Scalable Coherence Interface Draft 2.0*, March 1992.

[11] Ron Minnich, Dan Burns, and Frank Hady. The Memory-Integrated Network Interface. *IEEE Micro*, 15(1):11–20, February 1995.

[12] Andreas G. Nowatzyk, Michael C. Browne, Edmund J. Kelly, and Michael Parkin. S-connect: from networks of workstations to supercomputer performance. In *Proceedings of the 22nd International Symposium on Computer Architecture (ISCA), Santa Margherita Ligure, Italy*, pages 71–82, June 22-24 1995.

[13] Knut Omang. Performance results from SALMON, a Cluster of Workstations Connected by SCI. Technical Report 208, University of Oslo, Department of Informatics, November 1995.

[14] Scott Pakin, Mario Lauria, and Andrew Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, San Diego, California, December 3-8 1995.

[15] Peter Ross. Unix ^{TM} clusters for technical computing. Technical report, Digital Equipment Coropration, December 1995.

[16] Thorsten von Eicken, Anindya Basu, and Vineet Buch. Low-Latency Communication Over ATM Networks Using Active Messages. *IEEE Micro*, 15(1):46–53, February 1995.