# The ParaPC / ParaStation Project: Efficient Parallel Computing by Clustering Workstations

Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy
University of Karlsruhe, Dept. of Informatics
Postfach 6980, D-76128 Karlsruhe, Germany
email: {warschko,blum,tichy}@ira.uka.de

Technical Report 13/96

## Abstract

*ParaStation is a communications fabric for connecting off-the-shelf workstations into a supercomputer. The fabric employs technology used in massively parallel machines and scales up to 4096 nodes. The message passing software preserves the low latency of the fabric by taking the operating system out of the communication path, while still providing full protection.*

*The first implementation of ParaStation using Digital's AlphaGeneration workstations achieves end-to-end (process-to-process) latencies as low as 2.5$\mu s$ and a sustained bandwidth of more than 10 MByte/s per channel with small packets. Benchmarks using PVM on ParaStation demonstrate real application performance of 1 GFLOP on an 8-node cluster.*

## 1 Introduction

Networks of workstations and PCs offer a cost-effective and scalable alternative to monolithic supercomputers. Thus, bundling together a cluster of workstations – either single-processors or small multi-processors – into a parallel system would seem to be a straightforward solution for computational tasks that are too large for a single machine. However, conventional communication mechanisms and protocols yield communication latencies that make only very large grain parallelism efficient. For example, typical parallel programming environments like PVM[BDG$^+$93], P4[BL92] and MPI[CGH94] have latencies of several milliseconds. As a consequence, the parallel grain size necessary to achieve acceptable efficiency has to be in the range of tens of thousands of arithmetic operations.

In contrast, massively parallel systems (MPPs) offer an excellent communication/computation ratio. But engineering lag time causes a widening gap to the rapidly increasing performance of state-of-the-art microprocessors and low-volume manufacturing results in a cost/performance disadvantage. This situation is not unique to MPP systems; it applies to multiprocessor servers as well[ACP95].

ParaStation's approach is to combine the benefits of a high-speed MPP network with the excellent price/performance ration and the standardized programming interfaces of conventional workstations. Well-known programming interfaces ensure portability over a wide range of different systems. The integration of a high-speed MPP network opens up the opporunity to eliminate as much communication oberhead as possible.

The retargeted MPP-network of ParaStation was originally developed for the Triton/1 system[HWTP93] and operates in a 256 node system. Key issues of the network design are based around autonomous distributed switching, hardware flow-control at link-level, and optimized protocols for point-to-point message passing. In a ParaStation system, this network is connected via PCI-bus interface boards to the host systems. The software design focuses at standardized programming interfaces (UNIX-sockets), while preserving the low latency and high throughput of the MPP-network. ParaStation implements operating system functionality in user-space to minimize overhead, while providing the protection for a true multiuser/multiprogramming environment.

The current design is capable of performing basic communication operations with a total process-to-process latency of just a few microseconds (i.e., 2.5$\mu s$ for a 32bit packet). Compared to workstation clusters using standard communication hardware (e.g., Message-passing software such as PVM using Ethernet/FDDI hardware), our system shows performance improvements of more than two orders of magnitude on communication benchmarks. As a result, application benchmarks (i.e., ScaLAPACK equation solver and others) execute with nearly linear speedup on a wide range of different problem sizes.

## 2 Related Work

There are several projects targeting low-latency and high throughput parallel computing on workstation clusters.

MINI (Memory-Integrated Network Interface) [MBH95] targets a 1-Gbps bandwidth with 1.2 $\mu s$ latency interconnect using an ATM network. Communication in MINI is based on Channels between participating processes using ATM's virtual channel concept. Performance figures – ATM cell round-trip time of 3.9 $\mu s$ at 10Mbytes/s – are based on VHDL simulations; hardware development is in progress.

SHRIMP (Scalable High-Performance Really Inexpensive Multiprocessor) [BDF$^+$95] supports virtual-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. Using Pentium PCs as platform, the network interface is connected to an EISA-Bus (SHRIMP-I) and the Xpress memory bus (SHRIMP-II). A 16-node (SHRIMP-I) and a 2-node prototype (SHRIMP-II) was expected to be operational in 2Q/95.

Myrinet [BCF$^+$95] is a new type of local area network based on technology used for packet communication and switching within massively parallel processors. Measured performance using Myrinet API functions achieve one-way, end-to-end rates of 250 Mbps on 8-Kbyte packets.

Illinois Fast Messages [PLC95] is a high speed messaging layer that delivers low latency and high bandwidth for short messages. On Myrinet-connected SPARCstations, one-way latencies of 25$\mu s$ were meassured for small packets and for large packets, bandwidth as large as 19.6 MByte/s was achieved.

Von Eicken et al. adapted Active Messages [vECGS92, vEBB95] to a Sun workstation cluster interconnected by an ATM network. The prototype implementation shows a peak bandwidth of 7.5 MByte/s and a round-trip latency of 52 $\mu s$.

The Berkeley NOW (Network of Workstations) project [ACP95] targets 100+ workstation clusters using off-the-shelf components. One initial prototype is a cluster of HP9000/735s using an experimental Medusa FDDI interface. The final demonstration system will use either a second-generation ATM LAN or a retargeted MPP network, such as Myrinet.

Digital's MemoryChannel [Ros95] is a low latency cluster interconnect and provides a shared memory space among interconnected systems. It achieves a hardware latency of 5$\mu s$ and an aggregate bandwidth of 100 MByte/s on an 8-port hub.

Sun's S-Connect [NBKP95] is a high speed, scalable interconnect system that has been developed to support networks of workstations to share computing resources. In the S3.mp distributed, shared memory multiprocessor [NAB$^+$94], S-Connect switching fabrics deliver more than 100 MByte/s user programm accessible bandwidth at latencies of about 1$\mu s$.

ATM as fast workstation interconnect promised high bandwidth links as well as low network latency. With vendor supplied device drivers, however, end-to-end latency for small packets is worse over ATM than over Ethernet [KAP95, BBVvE95].

In contrast to most other approaches, we focus on a pure message passing environment rather than a virtual shared memory. As von Eicken at al. pointed out [vEBB95], recent workstation operating systems do not support a uniform address space, so virtual shared memory is difficult to maintain. Common to Active Messages and Fast Messages, performance improvement is based on user-level access to the network, but in contrast to them, we provide multiuser/multiprogramming capabilities. Like Myrinet, 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 Performance Hurdles in Workstation Clusters

Using existing workstation clusters as a virtual supercomputer suffers from several problems related to standard communication hardware, traditional approaches in the operating system, and the design of widely used programming environments.

Standard communication hardware (i.e.: Ethernet, FDDI, ATM) was developed for a LAN/WAN environment rather than MPP-communication. Network links are considered unreliable, so higher protocol layers must detect packet loss or corruption and provide retransmission. Common network topologies, such as a bus or a ring, do not scale very well. Sharing one physical medium among the connected workstations results in severe bandwidth limitations. A fixed or inappropriate packet size wastes bandwidth when transmitting small messages.

Common operating systems define two standardized interfaces: one to the hardware at device driver level and the other to the user as system calls. System calls provide the necessary protection needed in a multiuser environment. The device driver level provides a transparent interface for different hardware. This structure results in the inability to use specialized features of the communication hardware. As the communication subsystem of the operation system has to deal with different hardware abilities, the overall service is limited to the least common denominator. Thus there is no advantage for specialized communication hardware. Since the communication hardware is usually controlled by the operating system, sending or receiving a message implies at least one system call and copying the message buffer between user- and kernel-space. This takes more time than transmitting the message, especially for small messages. It also leads to large latencies and low throughput. Furthermore, built-in communication protocols (i.e. TCP/IP) are designed to support communication in local and wide area networks and therefore they are not very well suited for the needs of parallel computing. Their complex protocol stacks tend to limit the throughput of the whole communication subsystem.

Popular message-passing environments like MPI, PVM, P4 [CGH94, BDG$^+$93, BL92] and others, are usually build on top of the operating system interface. Many of them offer several parallel applications per processor. Running several parallel application on a workstation cluster results in scheduling and synchronization delays. The operating system of each individual node is only able to use local information to schedule processes, where global information would be appropriate. Thus, execution time of each application takes much longer than running the applications one after another. Running multiple processes of one application on one processor causes an additional process switching overhead, especially if these processes communicate with each other. Allowing several threads per parallel process forces the message-passing library – in addition to the operating system – to handle the relationship between incoming messages and associated threads. Furthermore, all of these environments allow processes to consume messages in arbitrary order and at arbitrary times. This implies message buffering at the destination node, unless the receiving process is setup for receipt.

## 4 The ParaStation Architecture

The ParaStation architecture is centered around the reengineered MPP-network of Triton/1 [PWTH93, HWTP93]. The goal is to support a standard, but efficient programming interface like UNIX-sockets. The ParaStation network provides a high data rate, low latency, scalability, flow control at link-level, minimized protocol, and reliable data transmission. Furthermore, the ParaStation network is dedicated to parallel applications and is not intended as a replacement for a common LAN and therefore eliminates the associated protocols. These properties allow using specialized network features, optimized point-to-point protocols, and controlling the network at user-level without operating system interaction. The ParaStation protocol 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 accessible message area. Within the ParaStation network protocol, operating system interaction is completely eliminated, removing it from the critical path of data transmission. The missing functionality to support a multiuser environment is realized at user-level in the ParaStation system library. This approach trades speed for inter-task protection.

### 4.1 Software Architecture

To avoid operating system overhead, all interfacing to the hardware is at user-level (see figure 1). The device driver is used at system-startup to configure the communication boards and within the startup-code of an application program to get some information about the hardware. During normal operation (i.e. message transfers) the ParaStation system library interfaces directly to the hardware without any use of the operating system. The necessary protocols to handle packet transmissions through the ParaStation network are mainly implemented in hardware. The gap between hardware capabilities and user requirements is bridged within the ParaStation system library.

The ParaStation system library (see figure 2) consists of three building blocks: the hardware interface layer, the central system layer, and the standardized user-interface (sockets).

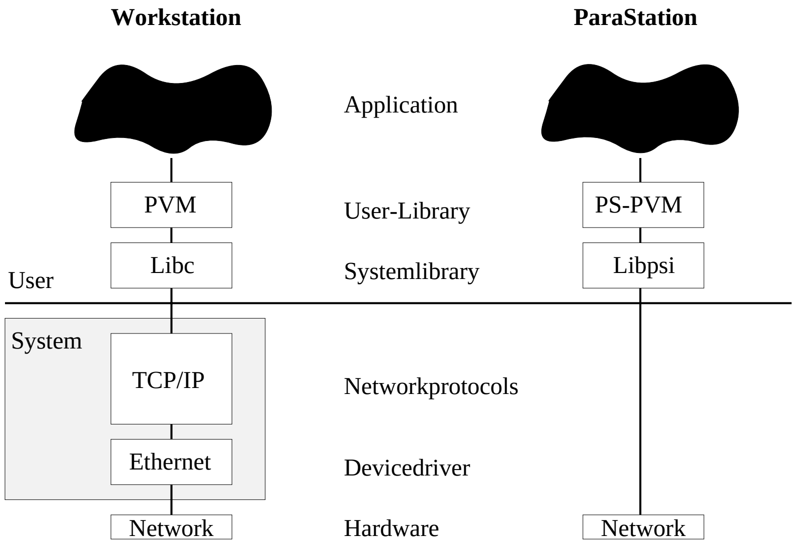


Figure 1: Difference between traditional network interfacing and the ParaStation solution

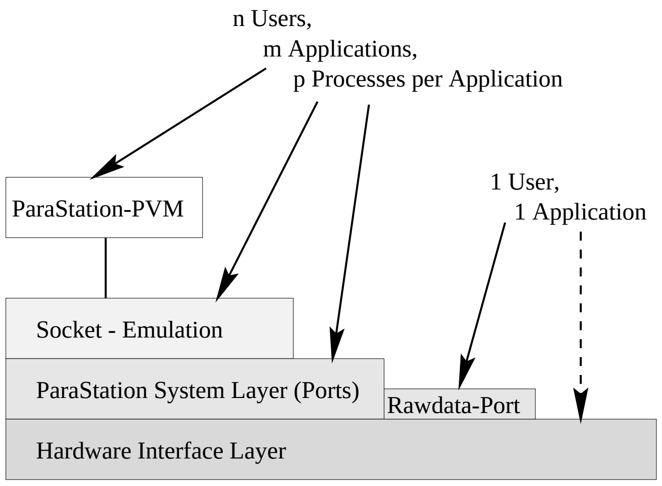


Figure 2: ParaStation system library

### 4.1.1 Hardware Interface Layer

The hardware layer provides an abstraction of the underlaying hardware. It is normally only used by the ParaStation system layer. The implemented functionality of 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. The initialization call is used for mapping 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. While sending a message, data is copied directly from user-space memory to the interface board and the receiving function does the same thing vice versa, eliminating all intermediate buffering. As a consequence, multiple applications using this layer are not supported. Nevertheless it is possible (but error prone) to use this layer directly as application interface.

### 4.1.2 System Layer (Ports)

The system layer provides the necessary abstraction (multiple communication channels) between the basic hardware capabilities (the hardware just handles packages) and a multi-user, multi-programming environment. Therefore we had to reassemble operating system functionality at user level to meet our primary design goal of efficiency.

This approach has several problems: First, the relationship between node addresses and individual communication channels has to be maintained. Second, mechanisms for mutual exclusion of critical sections have to be provided to ensure correct interaction between competiting processes. Third, fragmentation and reassembly of arbitrary sized messages without causing deadlocks is necessary to provide a suitable programming interface.

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-id's for different processes. Since message reception is done in user-space and at least the protocol information has to be received to get the destination-port of this 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 accessible message area to buffer this kind of messages. Maintaining 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 to prevent the network from 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.

The resulting implementation of these concepts contain not a single system call. Furthermore, we provide a zero-copy behavior (no buffering) whenever possible. To prevent deadlock situations, one buffer operation of a message to the message area is necessary. This leads to high bandwidth and low latencies.

#### 4.1.3 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. The interface can even handle file access when using `read/write` calls instead of `send/recv`. `Send/recv` calls, which can be satisfied within the ParaStation-cluster do not need any interaction with the operating system.

#### 4.1.4 Application Layer

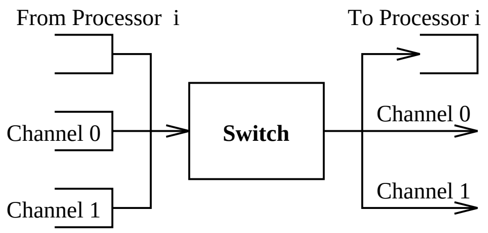
ParaStation implementations of standard programming environments like PVM [BDG+93], P4 [BL92], TCGMSG [Har91], and others use ParaStation sockets for high-speed communication. This approach allows us, to easily port, maintain, and update these packages. We use the standard workstation software.

The structure of the ParaStation system library provides well known interfaces (UNIX-sockets, PVM) to guarantee as much portability between different systems as possible, as well as low-latency, maximum-throughput interfaces (raw-data port, hardware layer) to get maximum performance out of the hardware.

### 4.2 Hardware Architecture

As communication hardware, we use the reenginneered MPP-network of Triton/1.

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 with three input buffers: 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 of 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.



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. First a packet is routed along the x-axis of the grid until it reaches it's destination column. Then it is routed along the y-axis to it's final destination node. Providing similar insertion rules as in the ring routing for both dimensions and giving the y-axis priority over the x-axis prevents deadlock.

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 (about 30 feet).

### 4.2.1 ParaPC Prototype

Prior to the ParaStation-System, we build an evaluation prototype called ParaPC. This testbed consists of two EISA-Bus based Intel 486 PC's (33MHz and 50MHz) connected through two ParaPC interface boards. We use BSD/OS V2.0 (BSD 4.4 from BSDI) as operation system. The ParaPC interface board is a slight modification of a board used in the Triton/1 project. Modifications were simple, but only one of the two possible network links as shown above is available – although this is no limitation in a cluster with two workstations.

In a first step, we implemented the hardware interface layer as presented in section 4.1. Using carefully designed assembler routines and some EISA-Bus specific features (automatic 32 to 16 bit translation), communication latencies of 5 $\mu s$ and transfer rates up to 4.8MByte/s could be achieved. Bypassing the operation system in BSD/OS is very simple by using the `ioport` command to allow user-level access to all registers of the interface board. Further results are presented in section 5.

### 4.2.2 ParaStation

The new ParaStation hardware extents the capabilities of the ParaPC hardware in several ways.

First, the ParaPC interface board was based on the EISA-bus. This bus is no longer state of the art, so we redesigned the board for the PCI bus. PCI was chosen because it meets our throughput requirements and it is available in several systems from different vendors (Intel based systems, Digital's Alpha stations, IBM's PowerPCs, and Sun's UltraSparcs). Thus, ParaStation is not limitated to machines supplied by one specific vendor.

Second, the prototype board had only one of the two possible network links as shown above and therefore the network topology was limited to an unidirectional ring. ParaStation supports both links and can choose between a bidirectional ring and an unidirectional two-dimensional toroidal mesh as network topology.

Third, our experience with the ParaPC prototype showed, that parallel applications often uses barrier synchronizations to keep conflicting memory accesses in sequence. So we decided to add hardware support for fast barrier synchronization.

The first realization of the ParaStation-System is based on AlphaGeneration workstations from Digital Equipment running Digital UNIX (OSF/1) and ports to other platforms are on the way.

Our current testbed consists of two different ParaStation-clusters. One cluster is based on 21064A Alpha workstations (275 MHz, 64 MB memory) with 8 nodes. The other cluster consists of four 21066 Alpha workstations (233 MHz, 64 MB memory). All machines are running under Digital Unix 3.2c.

## 5 Benchmark Results

The benchmarks described in this section cover three different scenarios. The communication and synchronization benchmarks provide information about the raw performance of ParaPC and ParaStation. Although we call this raw performance, these benchmarks reflect 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 (see section 4.1). The third scenario deals with application performance, namely run time efficiency.

## 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 *PingPong* benchmark, process two does not wait for receipt of a message before transmitting. This is a more practical scenario for two processes exchanging messages.

```
Process1:                              Process2:
    measure start-time;                    measure start-time;
    DO i = 1,k                             DO i = 1,k
      send(message)                          send(message)
      receive(message)                       receive(message)
    ENDDO                                  ENDDO
    measure stop-time;                     measure stop-time;
    calculate latency and throughput;      calculate latency and throughput;
```

The following table contains the results from the *Pairwise Exchange* benchmark, while varying message size from 1 to 508 bytes[1]. Transmitting larger messages can be done by fragmentating them into several smaller packets. To get accurate timing information, we measured runtime of one million iterations ($k = 10^6$ in the above code fragment) for each packet size. For very short message sizes (word transfer), we use specialized routines with less overhead than the general block transfer routine.

| **ParaStation Performance** | | | | | |
|---|---|---|---|---|---|
| **Alpha 21064A, 275MHz** | | | **Alpha 21066, 233MHz** | | |
| Message size in bytes | Runtime per iteration in $\mu s$ | Throughput in MByte/s | Message size in bytes | Runtime per iteration in $\mu s$ | Throughput in MByte/s |
| Word transfer | | | Word transfer | | |
| 1 | 2.52 | 0.794 | 1 | 2.39 | 0.978 |
| 2 | 2.51 | 1.592 | 2 | 2.40 | 1.957 |
| 4 | **2.48** | 3.228 | 4 | **1.94** | 4.108 |
| 8 | 3.24 | 4.939 | 8 | 2.58 | 6.188 |
| Block transfer | | | Block transfer | | |
| 4 | 3.54 | 2.260 | 4 | 3.62 | 2.205 |
| 8 | 4.27 | 3.739 | 8 | 4.13 | 3.860 |
| 16 | 5.71 | 5.596 | 16 | 5.39 | 5.921 |
| 32 | 8.69 | 7.358 | 32 | 8.25 | 7.956 |
| 64 | 14.56 | 8.772 | 64 | 12.90 | 9.894 |
| 128 | 26.40 | 9.693 | 128 | 22.74 | 11.238 |
| 256 | 50.31 | 10.227 | 256 | 42.45 | 12.019 |
| 508 | 95.90 | **10.506** | 508 | 81.43 | **12.450** |

For small message sizes, ParaStation achieves transmission latencies (sending and receiving a message in user-space) as low as $2.5\mu s$ on systems with the 21064A processor and $1.9\mu s$ on systems with the 21066 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 system and $0.97\mu s$ for the 21066 system. For larger message sizes, when overhead per byte decreases, we get a total throughput of up to 10.5 MBytes/s (21064A) and 12.5 Mbytes/s (21066) respectively. The performance differences are due to the internal architectures of the processors. The Alpha 21066 has the PCI interface on chip, where as the Alpha 21064A is using a board-level chipset.

Even the EISA-bus-based ParaPC prototype achieved communication latencies as low as $5\mu s$ and a total throughput of 4.8 MBytes/s.

## 5.2 Synchronization Benchmark

As mentioned above, SPMD-style parallel programs often need barrier synchronizations to keep their processes in synchrony. The following code fragment was used to measure the performance of our hardware supported synchronization mechanism on ParaStation.

[1] 508 bytes user data is the maximum packet lenght of the ParaStation interface.

```
Process1:                              Process2:
    measure start-time;                    measure start-time;
    DO i = 1,k                             DO i = 1,k
      sync()                                 sync()
    ENDDO                                  ENDDO
    measure stop-time;                     measure stop-time;
    calculate latency and throughput;      calculate latency and throughput;
```

To get accurate timing information, we measured runtime of one million iterations ($k = 10^6$) of the shown code fragment. To compare our results to conventional methods, we also implemented a logarithmic barrier synchronization using standard operating system calls.

| | ParaStation | | Ethernet | |
|---|---|---|---|---|
| Number of stations | Runtime per iteration in $\mu s$ | Synchronizations per second | Runtime per iteration in $\mu s$ | Synchronizations per second |
| 2 | 1.6 | 625.000 | 576 | 1739 |
| 4 | 1.7 | 588.000 | 1223 | 818 |
| 8 | 2.3 | 435.000 | 1856 | 539 |

The performance improvement of our hardware mechanism shown in the table above is so overwhelming, that no further explanation is needed. The shown results were measured on the 21064 cluster, the 21066 cluster is about 17% faster.

## 5.3 Performance of the Protocol Hierarchy

Switching from single to multi programming environments often suffers from a drastic performance decrease. In the following table, performance figures of all software layers in the ParaStation system are presented. The benchmarks were executed on the 21064A (275 MHz) cluster.

| | ParaStation | | OS/Ethernet | |
|---|---|---|---|---|
| Protocol-layer | Latency [$\mu s$] | Throughput [MByte/s] | Latency [$\mu s$] | Throughput [MByte/s] |
| Hardware interface | 1.24 | 10.5 | | |
| Rawdata | 4.15 | 9.6 | | |
| Port | 8.85 | 8.9 | | |
| Socket | 11 | 8.8 | 283 | 0.99 |
| P4 | 108 | 7.5 | 344 | 0.95 |
| PVM | 246 | 6.7 | 613 | 0.84 |
| Socket (self) | 6.4 | 85 | 195 | 22 |

Our system-layer (ports) only needs 7.6$\mu s$ additional cost, to support a true multi programming environment and the loss of throughput is within 15%. Furthermore, our decision to maintain the rawdata port is justified by the shown results. The rawdata port is twice as fast in latency than regular ports and loss of throughput is about 8.5% compared to the performance of the hardware interface. 4.15$\mu s$ latency is even less than the time for a null system call (4.5$\mu s$) on this particular machine. Most of these 4.15$\mu s$ is used to guarantee mutual exclusion and correct interaction between competitive processes.

The real advantage of ParaStation becomes obvious when comparing the performance to regular operating system calls. ParaStation socket calls are about 26 times faster in latency than the regular OS calls, while offering the same services. 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, these environments add an overhead of factor 9.8 (P4) and 22 (PVM) to the latency of our system-layer. Even in the standard operating system environment, P4 adds about 21% and PVM 116% overhead. 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 meassure 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 reflects mainly the *memcopy* performance of the system.

## 5.4 Application Performance

Focusing only on latency and throughput 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[2] library [CDD+95], which first uses BLACS[3] [DW95] and then PVM as communication subsystem on ParaStation. On ParaPC, we use a parallelized version of LINPACK [Don95].

All ParaStation application benchmarks were executed on the Alpha 21064A (275 MHz) cluster.

The *heat diffusion* benchmark starts with an even temperature distributions on a metal plate. On all four sides different heat sources and heat sinks are asserted. 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 1000 iterations, visualizing the result every 20 iterations.

| **Heat diffusion on ParaPC** | | | |
|---|---|---|---|
| size (n) | time for 1 workstation [ms/iter] | time for 2 workstations [ms/iter] | speedup |
| 32 | 5.16 | 3.40 | 1.52 |
| 64 | 19.91 | 11.07 | 1.80 |
| 128 | 79.96 | 41.77 | 1.91 |
| 256 | 330.96 | 164.47 | 2.00 |
| 512 | 1371.30 | 699.15 | 1.96 |
| 1024 | 5526.37 | 2841.46 | 1.94 |

As expected, execution time on uniprocessor and multiprocessor configuration quadruples as problem size is doubled. This is obvious, because the asymptotic work of a Jacobi-iteration on a $n \times n$ matrix is $O(n^2)$. For a wide range of problem sizes (starting with n=128 up to n=1024), an ideal speedup and therefore an efficiency of more than 95% is achieved. Only when benchmarking very small problem sizes, we see a decreasing speedup because the overhead for collecting and visualizing the data is quite large compared to computational task within the Jacobi-iteration.

The following table presents the results for ParaStation. Each experiment was measured with at least 5000 iterations, visualizing the progress every 20 iterations.

| **Heat diffusion on ParaStation** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 workstation | 2 workstations | | 4 workstations | | 8 workstations | |
| Problem size (n) | Runtime [ms/iter] | Runtime [ms/iter] | Speedup | Runtime [ms/iter] | Speedup | Runtime [ms/iter] | 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 |

As shown, we archieve a reasonable speedup for relevant problem sizes on all configurations. Taking the last line as an example, the efficiency of two workstations is close to its maximum. In the four and eight processor configuration, we achieve an efficiency of 93.75% and 92.5% respectively. In general, there are only two points where performance decreases when switching to the next larger configuration. But this only happens for problem sizes where parallelizing is doubtful.

The *Linpack* benchmark is an equation solver using LU decomposition with partial pivoting and backsubstitution. We have parallelized the core routine (SGEFA). We use a cyclic distribution of lines to achieve an optimal load balance. Thus, pivot searching, scaling and row elimination can be done in parallel.

The following table shows runtime, achieved Mflops and effective speedup for different problem sizes.

[2]Scalable Linear Algebra Package.

[3]Basic Linear Algebra Communication Subroutines

| **LINPACK on ParaPC** | | | | | |
|---|---|---|---|---|---|
| size | 1 workstation | | 2 workstations | | speedup |
| (n) | time[s] | Mflop | time[s] | Mflop | |
| 100 | 0.39 | 1.76 | 0.22 | 3.14 | 1.78 |
| 200 | 2.96 | 1.80 | 1.57 | 3.45 | 1.92 |
| 500 | 46.1 | 1.82 | 23.1 | 3.62 | 1.99 |
| 1000 | 368 | 1.82 | 183 | 3.64 | 2.00 |
| 1500 | 1255 | 1.80 | 627 | 3.60 | 2.00 |

The measured performance in Mflops of the uniprocessor configuration is quite stable over the whole range of different problem sizes and compares well to results presented by J. Dongarra [Don95] for a 33MHz 80486. Using two workstations we obtain a perfect speedup (greater than 1.98) and therefore an efficiency close to the maximum for all relevant problem sizes.

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 straight forward. Major goals within the development of ScaLAPACK [CDD$^+$95] 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.

| **ScaLAPACK on ParaStation** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Problem | 1 workstation | | 2 workstations | | 4 workstations | | 8 workstations | |
| size (n) | Runtime [s] | MFlop | Runtime [s] | MFlop | Runtime [s] | MFlop | Runtime [s] | 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 |

The above table confirms scalability of performance while problem size as well as number of processors increases. The efficiency of the two, four, and eight processor clusters are 94%, 87%, and 77% respectively. Remarkable is that we get more than a Gigaflop for the 8 processor cluster. These are real measured performance figures and not theoretically calculated numbers. The last line 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, presented performance results compare 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[4].

[4]For further information, see `http://wwwipd.ira.uka.de/parastation` or `http://www.hitex.com/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 a port to Pentium PCs running Linux. PC's running Windows NT are scheduled and Alphas running either Linux or NT will follow. Finally, we plan to support MPI as a future standard directly within the ParaStation system layer. This will give MPI applications a performance boost over a socket-based MPI implementation. Besides MPI, Active Messages and Fast Messages respectively are considered as additional interfaces to the system layer.

## References

[ACP95] 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.

[BBVvE95] 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.

[BCF$^+$95] 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.

[BDF$^+$95] 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.

[BDG$^+$93] 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.

[BL92] Ralph Buttler and Ewing Lusk. *User's Guide to the p4 Parallel Programmimg System.* ANL-92/17, Argonne National Laboratory, October 1992.

[CDD$^+$95] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. Scalapck: A portable linear algrbra library for distributed memory computers – design issues and performance. Technical Report UT CS-95-283, LAPACK Working Note #95, University of Tennesee, 1995.

[CGH94] Lyndon Clarke, Ian Glendinning, and Rolf Hempel. The MPI Message Passing Interface Standard. Technical report, March 94.

[Don95] Jack J. Dongarra. The Complete Linpack Report. Technical report, University of Tennesse, January 95.

[DW95] J. Dongarra and R. C. Whaley. A user's guide to the blacs v1.0. Technical Report UT CS-95-281, LAPACK Working Note #94, University of Tennesee, 1995.

[Har91] R. J. Harrison. Portable tools and applications for parallel computers. *International Journal on Quantum Chem.*, 40:847–863, 1991.

[HWTP93] Christian G. Herter, Thomas M. Warschko, Walter F. Tichy, and Michael Philippsen. Triton/1: A massively-parallel mixed-mode computer designed to support high level languages. In *7th International Parallel Processing Symposium, Proc. of 2nd Workshop on Heterogeneous Processing*, pages 65–70, Newport Beach, CA, April 13–16, 1993.

[KAP95] Kimberly K. Keeton, Thomas E. Anderson, and David A. Patterson. LogP Quantified: The Case for Low-Overhead Local Area Networks. In *Hot Interconnects III: A Symposium on High Performance Interconnects, Stanford University, Stanford, CA*, August 10-12 1995.

[MBH95] Ron Minnich, Dan Burns, and Frank Hady. The Memory-Integrated Network Interface. *IEEE Micro*, 15(1):11–20, February 1995.

[NAB$^+$94] Andreas G. Nowatzyk, G. Aybay, Michael C. Browne, Edmund J. Kelly, Michael Parkin, W. Radke, and S. Vishin. S3.mp: Current status and future directions. In *Shared Memory Multiprocessor Workshop. International Symposium on Computer Architecture, Chicago, IL*, May 1994.

[NBKP95] 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.

[PLC95] 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.

[PWTH93] Michael Philippsen, Thomas M. Warschko, Walter F. Tichy, and Christian G. Herter. Project Triton: Towards improved programmability of parallel machines. In *26th Hawaii International Conference on System Sciences*, volume I, pages 192–201, Wailea, Maui, Hawaii, January 4–8, 1993.

[Ros95] Peter Ross. Unix $^{TM}$ clusters for technical computing. Technical report, Digital Equipment Coropration, December 1995.

[vEBB95] 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.

[vECGS92] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19st Annual International Symposium on Computer Architecture*, Gold Coast, Australia, May 1992.