# PSPVM: Implementing PVM on a high-speed Interconnect for Workstation Clusters

Joachim M. Blum, Thomas M. Warschko, and Walter F. Tichy

University of Karlsruhe, Dept. of Informatics
Postfach 6980,76128 Karlsruhe, Germany
email:{blum,warschko,tichy}@ira.uka.de

**Abstract.** PSPVM in an implementation of the PVM package on top of ParaStations high-speed interconnent for workstation clusters. The ParaStation system uses user level communication for message exchange and removes the operating system from the critical path of message transmission. ParaStations user interface consists of a user-level socket emulation. Thus, we need only minor changes to the standard PVM package to get it running on the ParaStation system.
Throughput of the PSPVM is increased eight times and latency is reduced by a factor of four compared to regular PVM. The remaining latency is mainly (88%) caused by the PVM package itself. The underlying sockets are so fast (25$\mu$s) that the PVM package is the limiting factor. PSPVM offers nearly the raw performance of the network to the user and is object-code compatible to regular PVM. As a consequence, we achieve an application speed-up of four to six over traditional PVM using regular ethernet on a cluster of workstations.

## 1 Introduction

The PVM package is available on a wide range of parallel machines. These machines cover dedicated parallel machines as well as networks of workstations. On many dedicated parallel machines the supplied PVM uses proprietary send/receive calls to speed up communication. On networks of workstations all communication is usually done via the Unix socket interface. Socket communication relies on operating system functionality and common network protocols and is therefore limited in its speed. As long as these implementations were based on slow sockets the time lost in the rest of the PVM package (e. g. slow list implementation, packing/unpacking) was not the limiting factor. In this paper we present user level sockets on top of a high-speed interconnect where the speed is so fast that the PVM package limits the performance of the whole system.

## 2 Related Work

The public domain PVM package supports various multiprocessor environments. All of these are either based on shared memory multiprocessors (SunMP, SGI) or on message-passing multiprocessors (iPSC/860, Paragon, CM5). Common

to all is that there is a daemon running on a host computer. This daemon organizes the communication with the other systems in the virtual machine and allocates/deallocates compute nodes on the local parallel machine.

A hardware configuration comparable to the ParaStation is the IBM SP2. It is built by bundling together regular RS/6000 workstations in a 19-inch rack and connecting them with a high-speed communication switch. Each node is controlled by its own instance of the AIX operating system. The interface communicates in user space or in system space. Only one process per node can communicate in user space concurrently. IBM has implemented a specially adopted PVM version for the SP2 which is called PVMe[BR92]. This reimplementation is missing the powerful dynamic task creation of regular PVM. Crays PVM version for the T3D is also missing dynamic spawning of new tasks. Both implementation use special hardware features to speed up communication.

## 3 The ParaStation System

The ParaStation system is based on the retargeted MPP network of Triton/1 [PWTH93, HWTP93]. The goal is to offer MPP-like communication performance while supporting a standard, but efficient programming interface such as UNIX sockets. The ParaStation network provides a high data rate, low latency, scalable topologies, flow control at link level, minimized protocols, and reliable data transmission. It is dedicated to parallel applications and is not intended as a replacement for a common LAN. These properties allow the use of specialized network features, optimized point-to-point protocols, and controlling the network at user-level without operating system interaction.

The ParaStation system library provides multiple logical communication channels on one physical link. Multiple channels are essential to set up a multiuser/multiprogramming environment which is needed to support various interfaces such as Unix sockets. Protocol optimization is done by minimizing protocol headers and eliminating buffering whenever possible. Within the ParaStation network protocol, operating system interaction is completely eliminated, removing it from the critical path of data transmission. The functionality to support a multiuser environment is realized at user-level in the ParaStation system library.

### 3.1 ParaStation System Library

During normal operation the ParaStation system library interfaces directly to the hardware without any interaction with the operating system (see figure 1). The device driver is only needed during system and application startup.
Since the hardware consists of a retargeted MPP network, many protocol features are already implemented in hardware. The gap between hardware capabilities and user requirements is bridged within the ParaStation system library (see figure 2) which consists of three building blocks: the hardware abstraction layer, the central system layer, and an application layer which uses the standardized user interface (sockets).

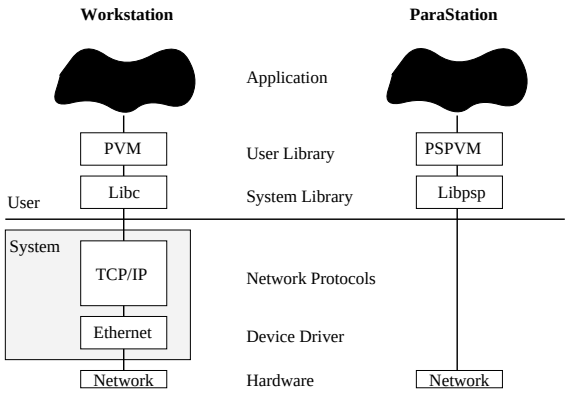


**Fig. 1.** Difference between traditional network interfacing and the ParaStation solution

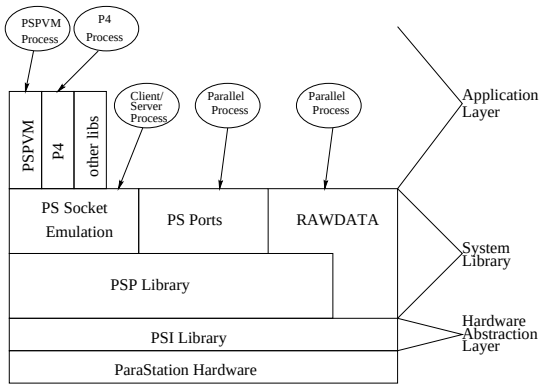


**Fig. 2.** ParaStation system library

**Hardware Abstraction Layer.** This layer provides an abstraction of the underlying hardware. It is normally used only by the ParaStation system layer. The implemented functionality of this layer consists of highly optimized send/receive calls, status information calls, and an initialization call 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. 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. This layer provides a true zero-copy protocol.

**System Layer (Ports).** The system layer provides the necessary abstraction (multiple communication channels) between the basic hardware abstraction layer capabilities and a multiuser, multiprogramming environment. We reassembled operating system functionality at user level to meet our primary design goal of efficiency.

To support individual communication channels (called *ports* in ParaStation) on top of the node-addressed hardware protocol, the system layer 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. For reasons of efficiency, semaphores necessary to ensure mutual exclusion while sending and receiving messages, are implemented at user level. 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.

The resulting implementation of these concepts contains no system call on the critical path of communication. Furthermore we tried to provide a zero-copy behavior (no buffering) whenever possible. To prevent deadlock situations, sometimes a single buffering is necessary. This technique leads to high bandwidths and low latencies.

**Socket Interface.** The socket interface 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. Porting an application is as easy as adding a prefix to all socket calls. For connections outside a ParaStation cluster, regular operating system calls are used automatically. 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.

**Application Layer.** ParaStation implementations of parallel programming environments such as PVM [BDG+93](see section 4), P4 [BL92], TCGMSG [Har91], and MPICH [GL] use ParaStation sockets for high-speed communication. This approach allows us to easily port, maintain, and update these packages. We can use the standard workstation distribution rather than reimplementing the functionality on our own.

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.

### 3.2 ParaStation Hardware

The ParaStation hardware [WBT96] consists of the retargeted and reengeneered MPP-network of Triton/1 [PWTH93, HWTP93]. 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 four buffers. The buffering decouples the network from local processing. The size of the packet can vary in the range from 4 to 508 bytes. Packets are delivered in order and no packets are lost. The network topology is either a two-dimensional toroidal mesh or a bidirectional ring. For both topologies a deadlock-free routing scheme is provided.

The ParaStation hardware is a PCI interface card. PCI was chosen because it meets our throughput requirements and it is available in several systems of different vendors (Intel-based systems, Digital's Alpha stations, IBM's PowerPCs, and Sun's UltraSparcs). The first implementation of the ParaStation system runs on Digital Alpha workstations with Digital UNIX (OSF/1); ports to other platforms are under way. At the time of this writing the first implementation on PCs running Linux is tested and shows excellent communication results.

## 4 PSPVM: ParaStation PVM

PSPVM is based on the ParaStation sockets. These sockets have the same functionality and same semantics as regular Unix sockets. The only two visible differences to the regular sockets are a prefix *pss* to each sockets call, and the higher speed of these sockets. Due to these sockets the changes to the standard library are small and an update to new versions of PVM is an easy task. We can use all advantages of the regular PVM package and communicate in user space without interaction with the operating system. Multiple processes are allowed on each node. The loopback of the ParaStation sockets is specially optimized (80MByte/s) and is mostly limited by the `memcpy` speed of the system.

An effect of a slow communication subsystem is that many programmers use *PvmRouteDefault* which sends a messages via the daemon on the same node which in turn contacts the daemon on the destination node, and delivers the message finally to the destination process. This routing scheme causes at least two process switches and two additional process-to-process latencies. Fast sockets reduce communication time so much that process switching and message transaction in the daemons are no longer a negligible part of the communication latency. To speed up communication, users should use the already provided *PvmRouteDirect* which establishes direct communication channels to the destination process.

The ParaStation system library supports most of the functionality of PVM and other message passing environments such as MPI and P4. A current project adjusts many programming environments to the ParaStation by using the ParaStation system library as a kernel. This approach enables us to only reimplement the environment-specific functionality and offer the whole speed of the hardware to the user.

## 5 Benchmark Results

The evaluation of PSPVM is done at two important levels. First, the latency and throughput of the ParaStation implementation is compared to the standard implementation. Second, an application benchmark is run on both implementations to compare the effect of an efficient communication subsystem to user applications. The tested ParaStation cluster consists of eight 21064A Alpha workstations (275MHz, 64MB memory). Our tests are all done with the same program object code, just linked with the other library.

### 5.1 Latency and Throughput of Sockets and PVM

Coarse-grained parallel programs depend only on throughput of the communication system, whereas fine-grained parallel programs also depend on low latency. The main reason that fine-grained parallel computation is inefficient on workstation clusters is that the latency of the communication subsystem is on the order of milliseconds. To measure the throughput und latency of the different communication subsystems we used a simple pairwise exchange benchmark. Both processes send and receive messages at the same time. This scenario is common in real applications. In the PVM versions of the program, packing and unpacking of the data is included. To get a measurement of the communication time and not of the packing, we used *PvmDataInPlace* buffer allocation.

| | **Throughput (in MByte/s)** | | | | **Latency (in $\mu$s)** | | | |
|---|---|---|---|---|---|---|---|---|
| Message | Standard | | ParaStation | | Standard | | ParaStation | |
| size | PVM | sockets | PSPVM | sockets | PVM | sockets | PSPVM | sockets |
| 4 | 0,01 | 0.01 | 0,04 | 0.31 | 733 | 566 | 200 | 25 |
| 16 | 0,04 | 0.05 | 0,17 | 1.21 | 748 | 631 | 191 | 26 |
| 64 | 0,16 | 0.19 | 0,61 | 3.53 | 764 | 655 | 204 | 36 |
| 500 | 0,69 | 0.70 | 3,16 | 8.34 | 1480 | 1411 | 316 | 119 |
| 2000 | 0,80 | 1.07 | 5,75 | 8.68 | 5073 | 3717 | 712 | 460 |
| 16000 | 0,79 | 0.89 | 6,41 | 8.73 | 41273 | 35816 | 5109 | 3663 |
| 64000 | 0,8 | 0.83 | 6,02 | 8.55 | 159682 | 153977 | 21214 | 14966 |

The PVM package adds about 200 $\mu$s to the latency caused by the underlying communication subsystem. This loss is not the dominating factor when built on top of regular sockets (loss of 29%), but it limits the performance when the system is built on top of ParaStation's user level sockets (loss of about 800%!!). Fine-grained parallel programs use many small messages and so these latencies dominate the performance of the whole application. On the other hand, the throughput of PVM is about 8 times higher on ParaStation. This improvement is achieved by just replacing the system calls by user level calls which use an additional PCI interface card. Therefore PSPVM has a substantial advantage even for coarse-grained programs. These numbers show that work must be done on the PVM system itself to speed up the message handling.

### 5.2 ScaLAPACK

The second application benchmark, *xslu*, taken from ScaLAPACK[1][CDD+95] is an equation solver for dense systems. Numerical applications are usually built on top of standardized libraries, so using these libraries as benchmarks is straightforward. ScaLAPACK is available for several platforms and presented results are directly comparable to other systems.

| **ScaLAPACK on ParaStation with PSPVM** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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** |
| **ScaLAPACK on Ethernet with PVM** | | | | | | | | |
| Ethernet | n=3000 | 165 | n=4000 | 232 | n=6000 | 320 | n=8000 | 261 |

The above table confirms scalability of performance in term of both problem size and number of processors. 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 on an 8-processor cluster. These are real measured performance figures and not theoretically calculated numbers. The last line shows the maximum performance one gets using ScaLAPACK configured with standard PVM (Ethernet). The best performance in this scenario is reached at a problem size of n=6000 on a 4-processor cluster. Using even more processors results in a drastic performance loss due to bandwidth limitation on the Ethernet. For ParaStation, in contrast, we see no close limitation when scaling to larger configurations. And it is even possible to improve the ParaStation performance by using a better interface than PVM.

## 6 Conclusion and Future Work

In this paper we have presented an efficient way of using PVM for fine-grained parallel programs on workstation clusters just by replacing the regular socket by user level sockets which use a parallel communication interface card. This change results in a much better performance of the whole system. We have experienced a speedup of three to four over regular PVM on a number of applications. It is also shown that the PVM library itself is now the limiting factor. To eliminate this bottleneck we plan to redirect PVM calls to the ParaStation system library, where most of the PVM functionality is already implemented. This reimplementation will be object-code compatible to regular PVM and will deliver the native

[1] Scalable Linear Algebra Package.

performance of the ParaStation environment. Most additional work will be done in an efficient implementation of allocating/deallocating PVM-specific buffers. Due to the PCI interface we are not limited to the DEC Alpha workstations and we are working on a port to Linux on PC/Alpha and ports to other platforms such as Windows NT on PC and Alphas are scheduled[2].

On the network side we target a 100MByte/s application-to-application throughput in a new-generation ParaStation board with fiber-optic links.

## References


[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 Lab., 1993.

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

[BR92] Massimo Bernaschi and Giorgio Richelli. PVMe: an enhanced implementation of PVM for the IBM 9076 SP2. In *ISCA*, 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.

[GL] William Gropp and Ewing Lusk. *User's Guide for* mpich*, a Portable Implementation of MPI.* Argonne National Laboratory.

[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.

[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.

[WBT96] Thomas M. Warschko, Joachim M. Blum, and Walter F. Tichy. The ParaPC/ParaStation project: Efficient parallel computing by workstation clusters. Technical report, University of Karlsruhe, Department of Informatics, March 96.


This article was processed using the LaTeX macro package with LLNCS style

[2] The information about the supported systems and many other useful information are available at `http://wwwipd.ira.uka.de/parastation`