EXPERIENCES WITH WINDOWS 95/NT AS A CLUSTER COMPUTING PLATFORM FOR PARALLEL COMPUTING.

MARKUS FISCHER * AND JACK DONGARRA †

Abstract. The increasing number of clusters built by commercial, off the shelf hardware shows a new trend in scientific computing. Within this movement is the propagation towards Windows NT (tm) as an operating system on PCs. The UNIX environment and Windows NT differ in terms of administrative issues as well as programming techniques. In this paper, we describe the modification of PVM (Parallel Virtual Machine) to interoperate with WIN32 (Windows NT 4.0 and Windows 95). PVM provides the functionality to accumulate the computing power of an NT cluster environment. Our migration from UNIX to the WIN32 architecture not only shows where porting existing software is easy and where more generic modules have to be designed, but also the limitations. PVM for UNIX and WIN32 environments is freely available at netlib: **http://www.netlib.org/pvm3/index.html**

Keywords: Heterogeneity, Message Passing, Cluster Computing, Networking, Windows NT

1. Introduction. Cluster computing on networks of workstations has been thoroughly researched on various UNIX-type architectures in the recent years. However, the familiar environment is about to change. Clusters built from commercial, off the shelf systems running Linux or Windows NT are joining the platform of scientific computing and are considered to be a new trend in supercomputing. Besides the inexpense of the hardware, its compatibility is a major advantage.

A major change can also be seen in the performance of commodity interconnects. New network technologies like Myrinet [2] or SCI [3] that connect single units to large scaling clusters offer communication latency and throughput previously known only on high-end, massively parallel systems.

PVM has been available for the WIN32 (Windows NT 4.0 and Windows 95) platform for approximately two years. In that time, the inverse relationship between price and performance of Intel based architectures has grown significantly. Thus it is not surprising to see an increased interest in clustering WIN32 machines using PVM as the distributed communication interface. Our statistics show that the download numbers of the WIN32 version make one third of the total downloads. NT is even being considered as an up-and-coming front-end standard. However, the WIN32 environment is not as practical for a cluster approach as one might think.

PVM has been the first message passing interface on top of Windows NT starting with PVM version 3.3.11 in May 1996. At this time MPI exists as well on Windows NT [5]. Different two these public domain versions, commercial software of PVM and MPI is also available. Calypso [7] is another possible solution for providing a parallel processing environment on Windows NT networks. Calypso, however, can only run within a homogeneous NT environment. PVM, on the other hand, runs in a heterogeneous environment across multiple platforms, which allows a virtual machine to consist of a variety of systems ranging from laptop's to Supercomputers. Besides different message passing implementation details, all projects have thus far originated from the UNIX world and have had to cope with Windows NT as a new environment. This migration can not be expected to be straight forward. Changing only the system calls does not work and several topics related to network computing have to be modified;

- security issues,
- the installation of additional services/daemons,
- interprocess communication, external data exchange
- dynamic process control for local or remote systems.

The subject of this paper is to evaluate the approach and feasibility of cluster computing on Windows NT. The paper is organized as follows. First we will give an overview of the PVM system followed by the description of our implementation. This will lead to a more general discussion about migrating from UNIX

*University of California, San Diego, CA 92093

†Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

to WIN32. Finally, after providing a small guide to our software package, we conclude our experience.

2. The PVM System. PVM (Parallel Virtual Machine) is a de facto standard message passing interface. It is an integrated set of software tools and libraries that emulate a general-purpose, flexible, heterogeneous, and concurrent computing framework on interconnected computers of varied architectures. The overall objective of the PVM system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation.

2.1. Available Architectures and Network Layers. PVM is available for **40+** different architectures combining Unix workstations, shared memory machines, and MPP's to one single parallel virtual machine. The WIN32 architecture was added to PVM in Version 3.4 after developing a compatible version to its predecessor. WIN32 is becoming popular and more important from day to day. This can also be seen in the amount of available hardware platforms. Following Intel and Alpha, SGI will support an Windows NT platform for their systems as well. Even though commercial processors achieve the same performance as processors on supercomputers at lower costs [6], the trend on local systems is the migration to SMP machines. As a matter of fact, the scheduling mechanism within Windows NT is being optimized for eight processors in the next versions. Multithreaded programming can then make use of the potential resources but can not perform distributed computing. Windows applications have existed for several years and the benefits that come with PVM are obvious. Using existing equipment and operating systems, PVM's application could be to

- solve large industry problems like parallelized combinatorical optimization algorithms (Job Scheduling , Cutting Stock) in your office
- use computers in High Schools for teaching purposes.
- combine built-in and compiler optimized functions with parallel processing power.
- route scientific data to standard front ends with special hardware (3D Graphic cards)

PVM's most powerful feature is that it provides the message passing interface that makes an application think its running on a single machine. Another advantage within the dynamic concept is that PVM is easily

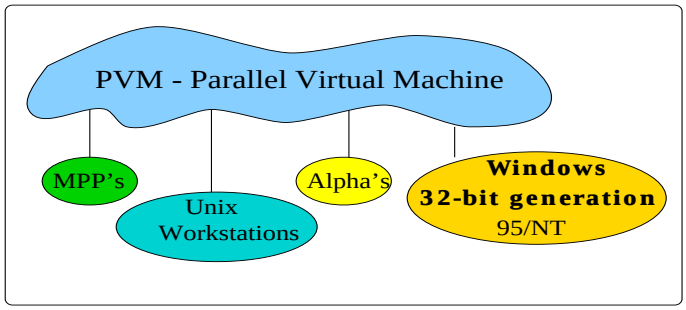

FIG. 2.1. *The PVM Model*

extendible to take advantage of other network architectures that are providing lower latency and higher bandwidth than conventional interconnects. UNIX extensions exist on Parastation [8] and SCI [4] to take advantage of the underlying network. SCI is also available on NT based on our implementation. A port of PVM on top of Easynet and BIP [9] is planned as well. Basically, a trend can be seen in eliminating the protocol stack when communicating with remote processes. The new network layers have in common that they offer mechanisms for bypassing the OS kernel, where convential protocols (using sockets for example) spend significant time. Such mechanisms reduce latency to 20-50 μ and increases bandwidth to 30 MB/s at the user level.

2.2. The WIN32 Implementation of PVM. Although Windows 9x and Windows NT are members of the WIN32 architecture, restrictions apply to the number of available WIN32 API calls for Windows 9x. While Windows 95 is designed to provide downgrade compatibility, NT is considered a real multiuser, multitasking operating system. As a matter of fact, only NT offers SMP capability. A further step towards

an WIN32 operating system is the separate process / address space in NT. This results in better performance and fault tolerance, whereas processes under Windows 9x only run in a virtual dos machine.
In prior versions of Windows, direct access to the hardware was possible which caused undesired shutdowns when conflicts arose. Windows NT is more stable since function calls are passed to device drivers, which access the hardware. This difference is kept transparent to the programmer. However, programming under security issues (which we will discuss later) as well as impersonation requires the use of different API calls. In this context, impersonation means the validation of new processes under a different account. As opposed to UNIX, this is not an easy task. Windows 95 allows only one user to be logged in at one time. Therefore, it does not provide a function which lets the system differentiate access privileges, though every process actually has administrative privileges. However, it is possible to allow multiple users to run their parallel applications on NT as well as on 9x. In general, one can say that WIN32 provides API calls that only work on one of the operating systems. Basically, Windows 9x only provides a subset of available WIN32 API calls. Because of this different functionality, the PVM code itself switches to the operating system being executed on. One major drawback with WIN32 is unavailable remote process creation. Unlike UNIX, basic services are missing in the standard WIN32 implementations. For remote startup, the PVM package provides an rshd compliant service that allows remote process startups after configuration. However, a username and the matching password have to be provided during service configuration, since both parameters have to be passed when calling an impersonating function. In this context, one might say that NT is not providing a friendly environment for distributed computation. However, future versions of WIN32 will provide mechanisms to allow remote computation more easily while also allowing users to see remote desktops, which is not supported yet either. Thus, it is not allowed to start up a GUI application on remote, since the application would pop up on a different user's desktop, which would likely be immediately killed.

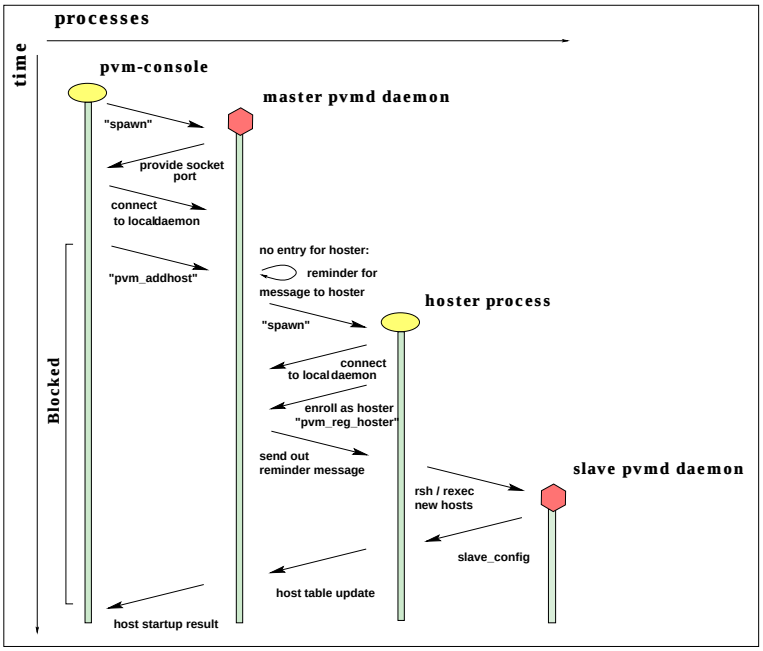

FIG. 2.2. *PVM Startup Protocol*

2.2.1. Process Control. Like the original PVM, this version uses the concept of daemon processes for the setup of the virtual machine. The main functions of these daemons are to start or delete new tasks, to route messages between tasks, and to establish direct connections for better performance [1]. Different from the UNIX version is the need for a separate *hoster* process, which is responsible for starting up slave daemons on other hosts. Invisible to the user, the host process is started automatically and runs as a task of

the virtual machine. Since PVM is designed to run accross multiple platforms, a common startup mechanism is required. Originating from UNIX the *rsh call* fires up remote daemons. Since PVM does not differentiate between hosts and architectures, a rsh call is also placed to add WIN32 hosts. To overcome the security problems involved with the installation of a rsh daemon it is also possible to perform a manual startup. Fig. 2.2 gives an overview of the startup of pvmd clients. It also portrays the processes involved when creating a virtual machine.

2.2.2. The Communication Layer. Messages between processes are exchanged using Windows WinSockets, which offer TCP and UDP on top of the IP layer. The specification is close to the BSD standard, however it is not possible to handle sockets as file or stream descriptors. It differs also from the standard by initializing and closing a socket structure, where the version of the socket layer must be specified. However, WinSockets allow communication across the WIN32 architecture and a faster TCP connection can be established as well. Using a heterogeneous system with different architectures requires encoding and decoding of messages if messages are sent to a different platform. The WIN32 version offers its own XDR en- / decoding possibilities. Like the existing PVM, message buffers can be created in three different manners: PvmDataInPlace (fastest method, only pointer to the message data is stored and packed for message transfer without encoding), PvmDataRaw (a copy of the data is made, sent out without encoding), PvmDataDefault (a encoded copy is sent out).
Messages sent between one architecture type should use PvmDataRaw or, even better, PvmDataInPlace, if possible. XDR en- /decoding is expensive and slows down the performance (dependent on CPU power).

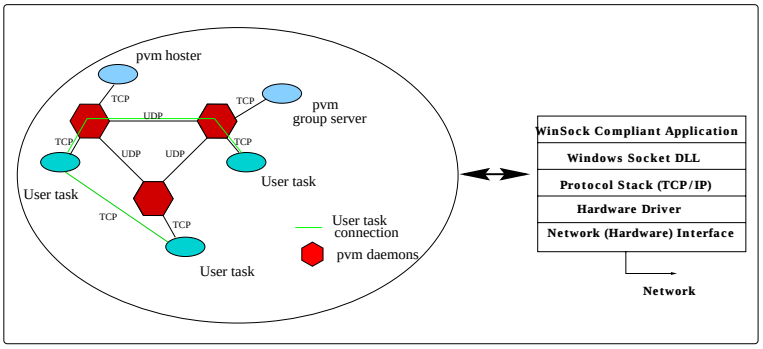

FIG. 2.3. *Communication and Layout of PVM*

2.2.3. Security Aspects. WIN32 offers sufficient security only for Windows NT. Security can be set to objects, where objects range from sockets, processes, files, pipes, or standard IO. This modification is done by the Security Identifier (SID), which exists for every user and in which the privileges of the specific users are set. In PVM, an important security aspect is a secure file system so that users can deny access to their files.
Unlike NTFS, Windows 95 does not offer a file system that restricts access. The FAT is readable for everyone. A process which would like to enroll into the virtual machine reads a socket address out of the `%PVM_TMP%\pvmd.username` file. Since this file is readable for every process, different users can get access to the pvmd daemon. For security reasons, it is therefore recommended to run NT as the operating system.

3. Using WIN32 as a Platform for Cluster Computing. When originating from the UNIX environment, a developer's first option is to look at the Windows NT POSIX subsystem. However, it only supports POSIX 1003.1, which was the only POSIX version standardized when Windows NT was created. Since then, there has been little demand for extending this subsystem because most applications have been converted to Win32. The 1003.1 system is of limited interest for fully featured applications because it does not include many capabilities (such as those in 1003.2, network support, etc.). Full featured applications that run under the Windows NT POSIX subsystem do not have access to Windows NT features available to

Win32 applications, such as memory-mapped files, networking, and graphics. Applications such as VI, LS, and GREP are the main targets for the Windows NT POSIX subsystem.
The second option UNIX programmers normally consider is using third-party, UNIX-like libraries to let their UNIX code compile as a Win32 executable. Several commercial (and at least one public domain) libraries do this. This is an option for some applications. The advantage of these porting libraries is that they minimize the initial porting effort. The main disadvantage for a competitive software product is that a native Win32 port of an application will generally be faster and will inevitably have more functionality. It can be awkward for the application to step outside of its UNIX shell if it needs to make Win32 calls to get more power from Windows NT.
The third option is porting UNIX applications directly to Win32. Using ANSI C/C++ libraries and commercial C compiler libraries, many of the traditional system calls UNIX applications rely on are available in Win32 applications.

3.1. Different Views of NT Clusters. Parallel computing on systems running a UNIX-like operating system brought along the ease of accessing different machines remotely. Access in this case means sharing computing resources as well as data storage. However, WIN32 is more likely a client server architecture in terms of file distribution except it does not offer remote activities at all. Besides offering remote computation, X Servers offer output routing of graphical applications running remotely to the machine where the user is actually interactively connected. Not only is this not possible across UNIX and NT, it is not even possible within the NT environment! The rsh-service provided by the PVM package adds an essential UNIX utility to WIN32. During installation, the user can choose between three installation types.
First is a *single node* type where PVM is installed on each drive of potential hosts. This requires entries to the registry as well as file storage on several hosts. The single node type, however, is not easy to maintain, since remote access is difficult. A registry for PVM_ROOT would look like PVM_ROOT=`c:\pvm\pvm3.4` for example. Second is a *server* type where PVM is installed on one sharable drive. This means that the executables must also be accessible from remote machines. This is fairly easy since WIN32 provides this *sharing* mechanism. Note that this access is granted to the file system only, not to computing resources. With this server type, a client type can be set up such that clients point their registry entries to the file location on the server. During startup, files would be transfered via the network. This type reflects the NFS installation known from UNIX. Clients would then have a PVM_ROOT entry of PVM_ROOT=`\\servername\c\pvm\pvm3.4`. Finally, a multiple Server type has to be used, when distributed applications are in different domains and a trust relationship is not set up for security reasons.

3.2. Extending current programming methods to more dynamic modules. Besides configuration management, the WIN32 architecture differs from UNIX-like environments in various ways when using it as a programming platform. Programming not only affects the call of different API's, but the modification of program layout is affected as well, since most of the known POSIX functions are not provided. Within the PVM on WIN32 project, first steps were made in porting an existing software package. This allows distributed computation to a different environment which further allows us to work on a more general specification of interfaces to an open, extendible environment. Portability is especially essential for HARNESS, our next, even more flexible, environment [11].
When choosing NT as a cluster computing platform, the following modules have to be redesigned or ported from UNIX:

WIN32 API Calls vs ANSI C

. The work involved in porting code is a question of how operating system-specific the existing application is. Basically, WIN32 provides a C Runtime library which covers most of existing application's code. But several functions or features are not available in WIN32 or have a different interface. Advanced programming using signals, for example, is not available under WIN32. Here, methods would have to be developed to use structured exception handling (SEH). Fortunately, threaded programming using *pthreads* is easily portable. Different API's have to be called for accessing or creating files under security. IO functionality is mimicked under WIN32 as long as the application is console-based. However, error handling has to be redesigned when the application is multithreaded. There are several other WIN32 API calls which are new when coming from

the UNIX site. Most of them are closely related to the operating system (like writing services, developing device drivers, creating several flavors of pipes) but they also deal with the versatile ways of configuring a machine. Depending on the purpose of a cluster (public pool, dedicated computing platform), one could lower security while restricting the environment to allow word processing only. Since there are many issues about migration that cannot be generalized, web sites or databases will help most in finding an individual solution (see [10] for example).

Security Issues
. Security in PVM is based on a secure file system. Convenient commands like chmod() are not available under WIN32. Whereas UNIX provides three classes of privileges (root, groups and user), WIN32 provides a variety of different privileges. Using a security model, such privileges can be assigned to handle objects. In this context, security can be set for file handling while restricting access to other users, even administrators. In this way, the security model in WIN32 is much better than in UNIX, however it is more difficult to program.

The Installation of Services/Daemons
. PVM uses rsh for starting up slave hosts. Since the PVM model is transparent to various architectures, it does not differ between architectures on startup either. Since WIN32 does not offer remote computation, a rshd compliant service had to be developed. During our research, we were surprised that, even with administrative privileges, impersonating a process is not easy. Processes need to have special privileges, which involves additional administrative work. In addition, the password has to be provided to impersonate the new process. ??The password is stored encrypted, however, questions the security when installing this service.??

IPC and Network
. WIN32 offers several mechanisms for IPC. In addition to anonymous pipes, it provides named pipes, which offer security even across the network. Named pipes, however, only works under NT. Other mechanisms are Mailslots or shared memory, which is based on the file interface yet kept in memory. These functions are not portable from UNIX but can be encapsulated in a module. Using sockets to communicate via the network requires additional startup, but as our experience shows, a fault tolerant wrapper is also needed around the send routines. While still up and running under WIN32, some sockets shutdown under UNIX. Besides this feature, WinSockets behave like expected and allow interoperability with UNIX without further complications.

Dynamic Process Control for Local or Remote Systems
. A major disadvantage is the lack of the fork() command. Several advanced UNIX programs use this call for convenient mechanisms. Some programs, however, follow the setting of new environment variables with an execute command. WIN32 offers several options to create a new task, but the CreateProcess() is most usable, since it enables setup of environment variables as well as special startup parameters (for debugging purposes for example). It is also the only command which has the same behavior under 9x or NT. Besides unimplemented functions in WIN32, it also offers calls which are not known from UNIX. One such call is CreateRemoteThread(), which might be useful for some applications.

Besides migration issues, WIN32 lacks standardization, which increases the amount of time in support. WIN32 is a highly commercial platform that is towing along a variety of software developers and vendors, which is causing the most problems when entering the new area.

3.3. Non - compatibility within WIN32. Our experience in migration reveals not only the need for dynamic modules in a plug-in like manner, but that the modules also have to cope with different compilers. Code running under Visual C++ might fail when recompiled with Watcom, for example. Even worse, libraries from different vendors can be used for linking. Therefore, we provide multiple compiler brands since a limitation to only one compiler would force users to purchase the provided product. Providing only the source files with a generic makefile that includes several compiler switches, as our experience shows, will result in a flood of help messages because NT offers too many methods of installation. But even after a successful compilation, the software may not work. Administrators may find their own method of installation

and put too tight of security on a system, which might not even allow the storage of temporary data for installation.

4. Using PVM. The hassle in providing basic information about how to set environment variables under WIN32 and the restrictions mentioned above resulted in providing a guided installation. Our product is using Installshield which is a Back, Next, and Browse-clickable tour to install PVM without any problems. Users can choose between a local, remote, or server installation. This installation is subdivided again in a *Typical, Common* or *Administrative* type where the latter one also allows the rshd configuration. Furthermore, different compiler libraries are provided and the registry is modified to start PVM with ease. For recompiling our examples, makefiles are modified during installation, so that the user is ready to go after clicking the *Finish* button.

```
c:> pvm hostfile
hoster() 2 to start
0. t80000 rudolph so=""
1. tc0000 thud so=""
Password
3.3.10
t40002
pvm> conf
3 hosts, 2 data formats
                HOST    DTID   ARCH   SPEED
                  ed   40000 WIN32    1000
             rudolph   80000  SUN4    1000
                thud   c0000  SUN4    1000

pvm> ps
                HOST    TID   FLAG 0x COMMAND
                  ed   40001 204/H,c c:/pvm/pvm3/bin/WIN32/hoster.exe

pvm> spawn -> spmd
1 successful
t80001
libpvm [t80001]: token ring done
pvm>
```

FIG. 4.1. *PVM Session*

4.1. Setting Up a Virtual Machine. The PVM console can be used as a convenient way to interact with the pvmd. This process can be used to add other hosts to the virtual machine. It will pass a hostfile to the master daemon if provided, as well as allow manual add's. It will also reset the parallel virtual machine if tasks hang.
Finally the PVM console provides the possibility of a proper shutdown. Tasks can be started within the console and the 'ps -a' command gives information about running tasks on every machine in the VM.

4.2. Creating and Debugging a PVM Application. Sequential code can be parallelized using PVM's message passing interface. The algorithm has to be changed so that processes can divide up work and gather the solution. These functions can be found in the PVM library.
A linking with `$(PVM_ROOT)`/lib/`$(PVM_ARCH)`/libpvm3.lib is necessary. When group functionality is needed, the application also has to link with `$(PVM_ROOT)`/lib/`$(PVM_ARCH)`/libgpvm3.lib.
Fortran applications will also run in WIN32. They have to be linked with
`$(PVM_ROOT)`/libfpvm/`$(PVM_ARCH)`/libfpvm3.lib. Note, that this is the first Fortran interface in a WIN32 message passing interface. Current WIN32 MPI implementations still do not support the Fortran interface. Debugging is also possible in the WIN32 version. The common way in PVM is to start a new task with

the PvmTaskDebug flag. Under UNIX the task was then started under a debugger. Different from the existing Unix version, the new task is started on the local or remote machine but waits for the connection of debugger. Users then have to start their debugger manually on the local machine and then choose the option for local or remote process attachment.

5. Results. We ran a benchmark program which performs a ping pong-type test between two processes. Each size of a message was sent out and received 100 times.

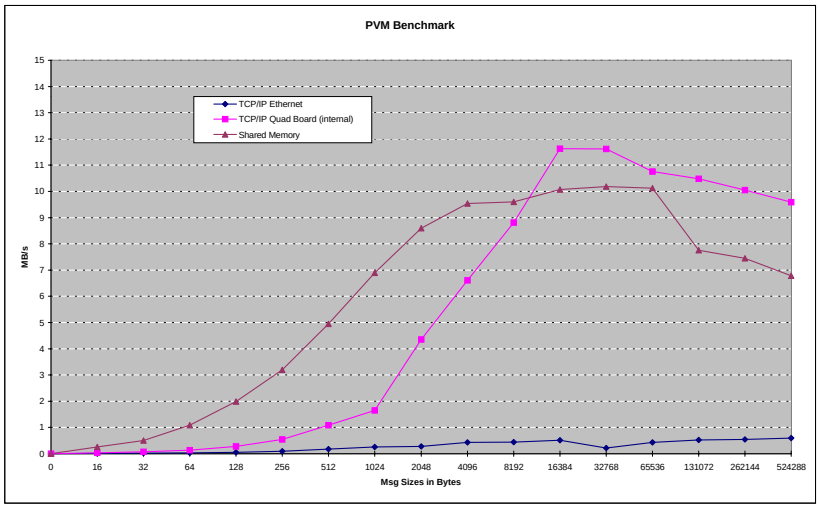

FIG. 5.1. *PVM Performance*

Based on these round trip values, an average round trip time is computed. Every message content was double checked on the receiver and on the initiator. The tests did not drop any messages and the messages were received 100 percent correctly. The internal TCP/IP communication achieves very high performance with up to 12MB/s, especially on multiprocessor boards.

6. Conclusion. As tests have shown, the PVM package on top of Windows NT/95 brings reliable parallel computing possibilities. Crashed applications do not interfere with the operating system and users do not have to to be afraid of causing unwanted shutdowns. It is also possible to have multiple, communicating processes running on one machine as well as share one host for multiple virtual machines. Users can take their existing PVM application and run it on the new architecture. There is no need for long lasting modifications. Basically, the number of extensions to new interconnects and the number of downloads occuring with WIN32 shows the popularity of PVM as a research platform.

REFERENCES

[1] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing* 1994. Published by MIT Press, Boston. http://www.netlib.org/pvm3/book/pvm-book.html

[2] Myricom. Myrinet link and routing speicifcation, 1995. http://www.myri.com/myricom/document.html

[3] IEEE Std for Scalable Coherent Interface (SCI). Inst. of Electircal and Electronical Eng., Inc., New York, NY 10017, IEEE std 1596-1992,1993.

[4] Markus Fischer, Jens Simon. *Embedding SCI into PVM. Proc. of EuroPVM/MPI '97*, Cracow, Poland, pp. 183-191,1997.

[5] M. Baker, G. Fox. *MPI on NT: A preliminary evaluation of the available systems* In Workshop PC-NOW, IPPS/SPDP98, Orlando, USA, 1998.
[6] Alfred Aburto *PDS: The Performance Database Server* November 26, 1995, Naval Ocean Systems Center, San Diego http://performance.netlib.org/performance/html/PDStop.html
[7] http://www.eas.asu.edu/ calypso/
[8] J.M. Blum, T.M. Warschko, W.F. Tichy. *PULC: ParaStation User-Level Communication. Design and Overview*, IPPS Orlando March 98.
[9] L. Prylli and B. Tourancheau. Bip: a new protocol designed for high performance networking on myrinet. In Workshop PC-NOW, IPPS/SPDP98, Orlando, USA, 1998.
[10] http://www.experts-exchange.com
[11] A. Geist et al. *Harness, the next generation beyond PVM.* http://www.epm.ornl.gov/harness