Another Architecture: PVM on Windows 95/NT

Markus Fischer [*,†] Jack Dongarra [*,‡]

October 4, 1996

Abstract

This paper describes the implementation of PVM in the new WIN32-bit world. There are no restrictions to existing applications which are using PVM since it is fully compatible to the existing PVM3 release. We discuss the limits and provide some benchmarking results. The software package is freely available at netlib: **http://www.netlib.org/pvm3/index.html**

1 Introduction

1.1 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 emulates a general-purpose, flexible, heterogeneous 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.

1.1.1 Current Architectures

So far, PVM is available for **40** different architectures combining Unix- workstations, shared memory machines and MPP's to one single parallel virtual machine. Obviously, the established architectures take place in the more scientific area.

1.1.2 New Architecture: Computers running WINDOWS 95/NT

However, since software companies like MS provide multiuser (WINDOWS NT) and multitasking (WINDOWS 95/NT) operating systems, lots of personal computers in smaller companies could be used for parallel processing if this processing power could be put together. As a matter of fact,

[*]Department of Computer Science, University of Tennessee, TN 37996
[†]Department of Computer Science, University of Paderborn, 33100 Paderborn, Germany
[‡]Mathematical Science Section, Oak Ridge National Laboratory, Oak Ridge, TN 37831

there are Unix -like operating systems (Linux for example) for PC's available, but they are not used by companies or individual people. It is Microsoft which leads the market with almost 90 percent. But not only improved operating systems and development tools lead to a growing market for scientific computing with PC's.
Figure 1 shows the growing performance of PC's compared to Unix workstations [PDS] (SUN workstations for example) which will make the use of more affordable PC's more attractive.

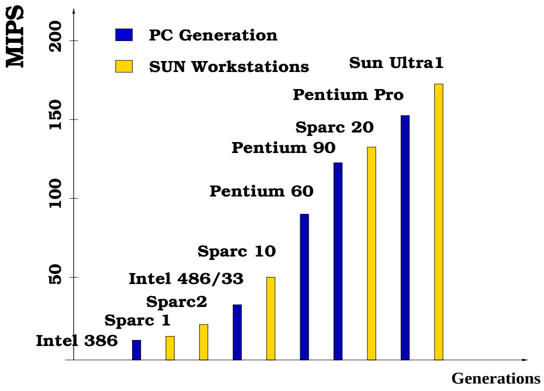

Figure 1: Growing Performance of PC's

Therefore, PVM now offers a version to this so called WIN32 - bit world. Using existing equipment and operating system, its application could be to

- solve large industry problems like parallelized combinatorical optimization algorithms (Job Scheduling , Cutting Stock) at 'home'.
- use computers in High Schools for teaching purposes. Basic steps in the more and more important becoming parallel processing area can be made.
- combine new WIN32 - compiler function calls (like built-in 3D rendering) with parallel processing power.

PVM's most powerful feature is that it provides the message passing interface which lets the application assume to run on one single machine.

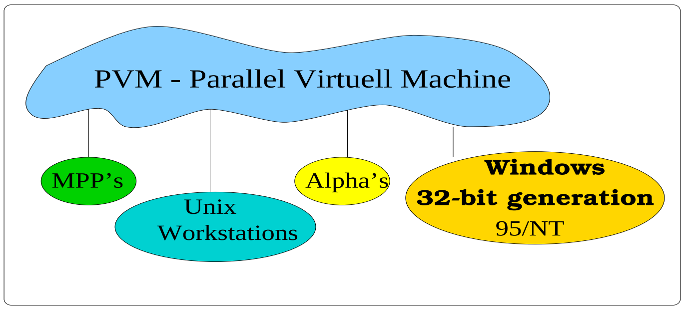

Figure 2: The PVM Model

2 Developing Applications in the WIN32-bit World

2.1 Towards scientific computing

A bit of history can let us understand our difficulties in programming for a parallel world for PC's. Microsoft's first operating system was its command-line MS-DOS. Because of the competition with Apple Macintosh the first version of Windows was released. It offered the first graphical user interface for PC's to ease working with computers. The more and more growing need of connected computers in companies leaded to Windows for Workgroups, followed by Windows 95/NT today. The newest version of these operating systems offer the same surface and are making use of the 32-bit availability.
Developing applications for both operating systems requires only one compiler. However lots of functions from the more professional Windows NT are not implemented in Windows 95. Basically it can be said that Windows 95 is a subset of Windows NT and is just on its way for becoming a real operating system.

2.2 Differences Between Windows 95 and Windows NT

The major difference is that Windows 95 is designed to provide downgrade compatibility. Programs developed under the early DOS 3.3 still run with Windows 95. Windows NT does not provide this MS-DOS shell. This is the reason why the PVM WIN32 executable itself **switches** to the running operating system.
The reason is the concept of using device drivers. User's code in Windows NT is not allowed to access hardware directly.
In former versions of Windows direct access was possible and caused unmeant shutdowns when conflicts arose. In Windows NT, function calls to device drivers, which actually access the hardware, guarantee that Windows NT runs stable (Process runstate level).
Another contrast to Windows NT is that Windows 95 allows only one user to be logged in at one time. It therefore does not provide a function which lets the system give different users different access privileges. Therefore every process actually has **'root'** privileges. Performance is also better on NT. It offers a Virtual Memory Space resulting in faster switching of multiple processes. Furthermore NT is designed to run as a multiple processor machine. Its scalability is dependant on the motherboard. Unfortunately we have no experience within that environment, however, it can be assumed that the operating system takes care of dynamic started processes and performs the mapping. This will probably lead to an investigation of a shared memory implementation on those systems.

2.2.1 Security Aspects

The new generation offers sufficient security only for the Windows NT side. Security is set to objects, where objects can be declared as multiple things. It is possible to handle a socket as an object, but also a process or a file, standard input, -output and -error can be seen as an object. Finally, a pointer to this object can modify the security status of this object. This modification is done by the Security Identifier, which exists for every user and in which the privileges of the specific users are set. In PVM an important security aspect is a secured file system in which users can deny access to this file to other users.

To the contrast of Windows NT's NTFS, Windows 95 does not offer a file system which restricts access. The FAT is readable for everyone. The reason was mentioned above, once logged in, the process has root privileges. A process which likes to enroll into the PVM reads a socket address out of the `%PVM_TMP%\pvmd.username` file. Since this file is readable to every process different users could get access to the pvmd daemon ! It is recommended to run NT as the operating system, however people are using Windows 95 more often than NT because of the downgrade capability.

3 The Implementation

Like the original PVM this version needs a daemon process called pvmd, which keeps track of the entire task management. The first pvmd process becomes a master pvmd. In the original version this process is also used to start up slave daemons on other hosts. This version however uses a separate hoster process. This is kept invisible to the user, so that he will not notice any difference. The hoster process is started automatically and is running as a task of the virtual machine. He also will be started again automatically, once a user has executed a reset.

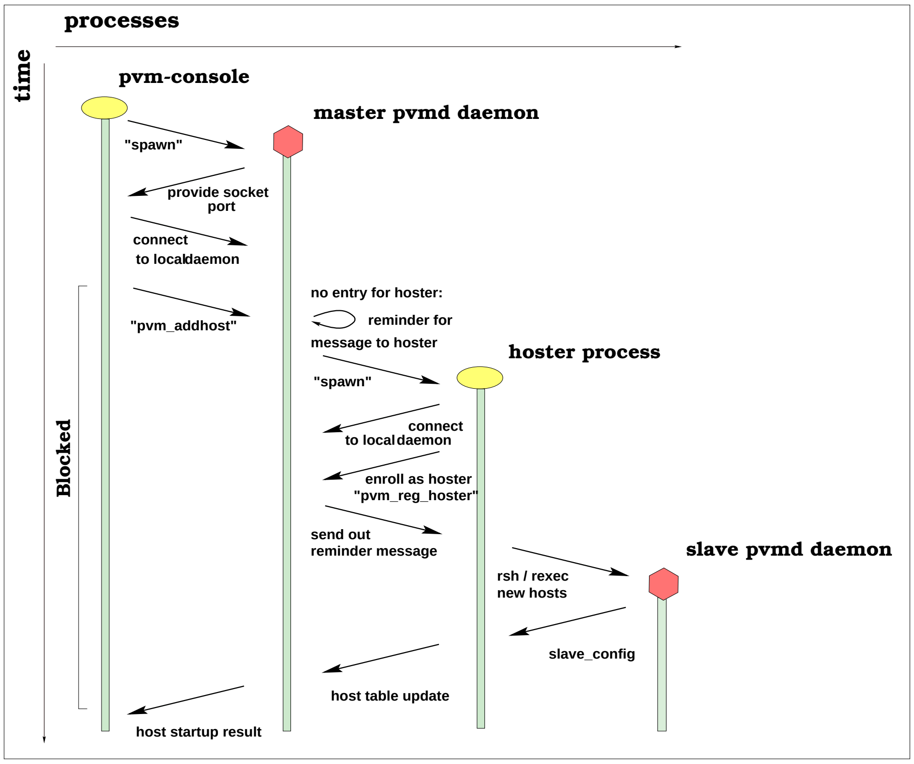

Figure 3: PVM Startup Protocol

The main functions of these daemons are to start or delete new tasks, to route messages between tasks, but also to establish direct connections for better performance. For more detailed information the interested reader may refer to the book about PVM [PVM].

3.1 The Communication Layer

Messages between processes are exchanged using Windows Winsockets. They 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 them as file or stream descriptors. It differs also from the standard by initializing a socket structure, where the version of the socket layer must be specified. A socket failure is reported otherwise.

Using a heterogeneous system with different architectures requires encoding and decoding of messages. The WIN32 version offers 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 inbetween one architecture should use PvmDataRaw or better PvmDataInPlace, if possible. XDR en- /decoding is expensive and slows down the performance (dependend on CPU power).

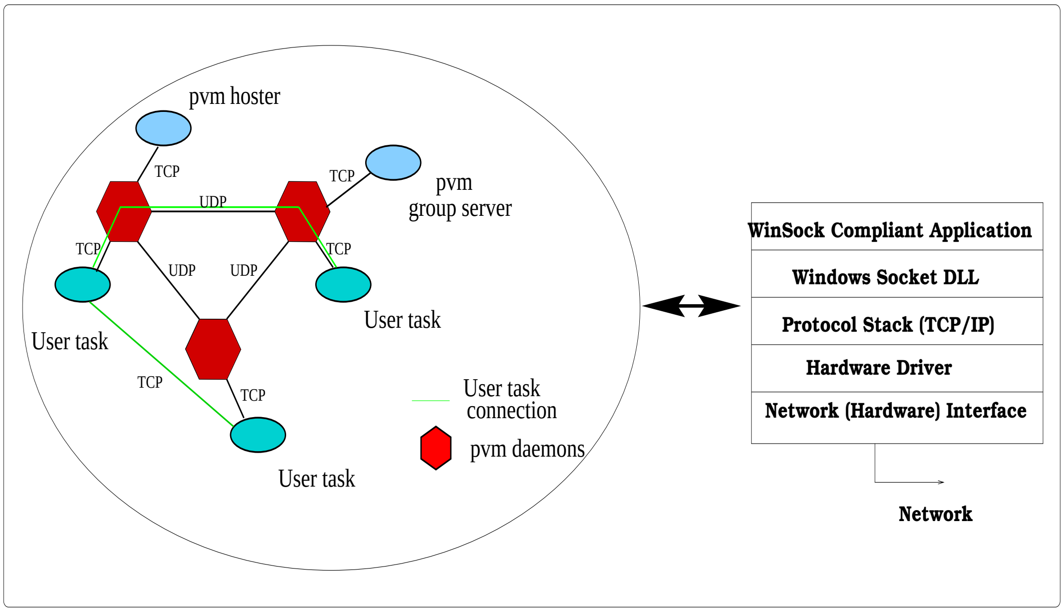

Figure 4: Communication and Layout of PVM

3.2 Differences Between WIN32 and UNIX

In the Unix environment a user has his unique user-id. Furthermore all users have the same rights concerning executing processes. The easy method of the Unix call chmod secures files from being read by other users. It can be allowed / denied to all users or the access may be restricted to a specific group. In WIN32 each user has to be included into a security object if he wants to have access to it. If a new object is created, a security attribute is generated to this object and if not specified otherwise, access is granted to everyone. Users even have special rights in this environment. More specific the user's SID, the Security Identifier, which is a binary structure

stores the rights for the user which are set by the administrator. The SID is invisible to other users and information can only be obtained by using specific function calls. Therefore different PVM's in the WIN32 environment are distinguished by there user name, which is also stored in the SID and is connected to the running process. Creating a new process can be done by several spawn calls which will generally take a filename as its argument. Different flavors provide environment setting or startup parameters. It is possible to start a new process but it is not possible to split up a task in the way the Unix 'fork' call does.
This leads to the need of a hoster process which is responsible for starting up new daemons on other hosts. If the master daemon would perform this startup, the process would block, waiting for a response or sending initial data to the new hosts. Consequently, other input would have to wait like the request of starting up new hosts, which is sent to the master daemon, routing messages or startin up new tasks, for example. The hoster method keeps the master daemon free to response to other requests.

4 Using PVM

Before we describe the software handling of this version we point to the additional requirements.

4.1 Settings

The correct use of PVM for WIN32 needs the following environment variables,

- PVM_TMP which specifies the location of the temporary files (PVM_TMP=`c:\temp`)
- PVM_ROOT points to the installation of pvm (PVM_ROOT=`c:\pvm\pvm3`)
- PVM_RSH locates the rsh-command (On NT: `%winntsystem%\system32\rsh.exe`)
- PVM_ARCH has to be set to WIN32

As well as Windows 95, Windows NT is designed for networking. Nevertheless they do not provide convenient tools for remote process handling. As a matter of fact at least one additional daemon has to run on each host. Users have to look out for a remote shell daemon (rshd), which will allow to add other hosts to the machine. If you have a different account on the other machine, you will probably need a remote execute daemon (rexecd). Note that Unix does provide those. They are only required on WIN32 computers. It is also possible to perform a manual startup. This, however, is very inconvenient and takes time.

4.2 Setting Up a Virtual Machine

To provide a convenient way for the user to interact with the pvmd, the pvm-console process can be used. Here new hosts can be added to the virtual machine and passwords can be typed in. The user can reset his parallel virtual machine if tasks hang. Last not least in provides the possibility of a proper shutdown. Figure 5 shows a typical startup of a virtual machine. After starting up the master pvm daemon via pvm (figure 5) the user can add other hosts to his virtual machine. (host 'rudolph' and 'shenzi' respectively). Tasks can be started within the console and the 'ps -a' command gives information about running tasks on every machine in the VM.

```
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   40000WIN32     1000
                 rudolph   80000  SUN4     1000
                    thud   c0000  SUN4     1000

pvm> ps
                    HOST    TID  FLAG 0x COMMAND
                      ed  40001204/H,c c:/pvm/pvm3/bin/WIN32/hoster.exe

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

Figure 5: PVM Session

4.3 Creating a PVM Application: FORTRAN and C

Sequential code can be parallelized using PVM's message passing interface. The algorithm has to be changed that processes can divide up work and gather the solution. These functions can be found in the library of PVM. A linking with `$(PVM_ROOT)`/lib/`$(PVM_ARCH)`/libpvm3.lib is necessary. The new PVM version also offers group functionality. Application which are using the group server have to link with `$(PVM_ROOT)`/lib/`$(PVM_ARCH)`/libgpvm3.lib, too. It is also possible to bring existing FORTRAN applications to the new environment. They have to be linked with `$(PVM_ROOT)`/libfpvm/`$(PVM_ARCH)`/libfpvm3.lib.

4.4 Debugging in PVM on WIN32

The common way in the existing PVM version was to start a new task under a debugger, which was specified in the `$(PVM_ROOT)`/lib/debugger file. This debugger was displayed at the users machine. Debugging is also possible in the WIN32 version. Different to the existing Unix version the new task is started on the remote machine, but waits for the connection to a debugger. Users then have to start their debugger manually on the local machine and then choose the option for remote connection. For developing, we advise to test the application on the local machine. The operating system allows local socket connections without restrictions. (See figure 6)

5 Results

We ran a benchmark program which performs a ping pong test between two processes. Each size of a message was sent out and received 100 times. 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 message and they were received correctly by 100 percent. The result for the WIN32 environment lacks good performance. We optimized the code and will

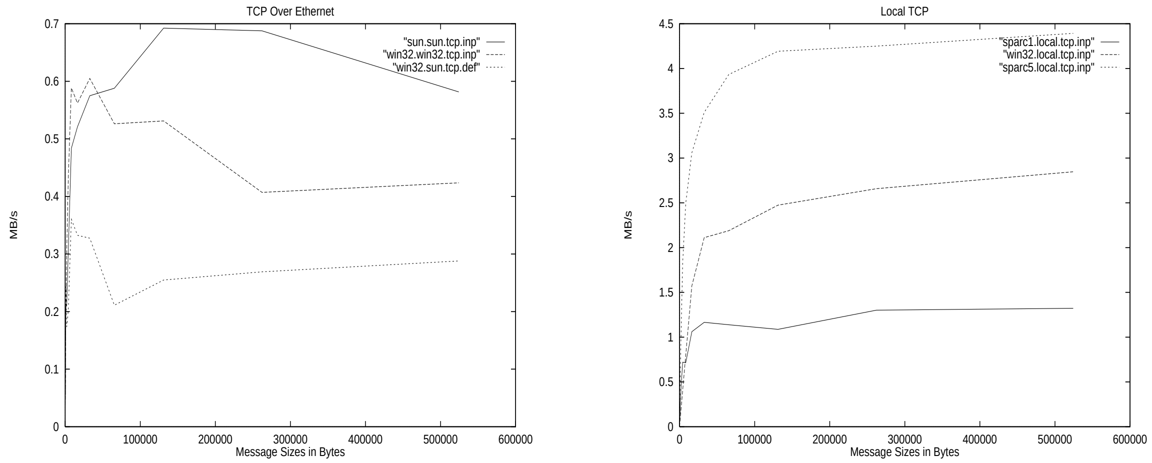

Figure 6: PVM Performance

include the new winsocket library (Winsocket 2 specification), which will lead to better values.

6 Conclusion

As tests have shown, the PVM package on top of MS 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 unmeant shutdowns. It is also possible to have multiple, communicating processes running on one machine.
The version to the WIN32 world was done by using MS VC++ compiler but the library can also be linked by other brands compilers.
Also a cross compiling of this package with fortran compilers was done successfully. Users can even take their existing PVM - application and run it on the new architecture. There is no need of long lasting modifications.
The only restriction is the possibility of using the `pvm_sendsig` call, which is usually provided by the PVM interface.

References

[PVM] 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

[PDS] Alfred Aburto *PDS: The Performance Database Server* November 26, 1995, Naval Ocean Systems Center, San Diego
http://performance.netlib.org/performance/html/PDStop.html