PVM Toolkit for Windows

Jan Kwiatkowski[1,2], Piotr Hojnor[2]
[2] Computer Science Department,
Wrocław University of Technology
Wybrzeże Wyspiańskiego 27
50-370 Wrocław, Poland
phone: (+48)(71) 3203602
fax: (+48)(71) 3211018
e-mail: {kwiatkowski,hojnor}@ci.pwr.wroc.pl

Cezary Janikow
[1] Math and Computer Science Department
University of Missouri – St. Louis
8001 Natural Bridge Rd,
St. Louis, MO 63121, USA
phone: (314)5166352
fax: (314)5165400
e-mail: janikow@arch.umsl.edu

Abstract: *In the past years, computing has been moving from the sequential world to the parallel one, from centralized organization to decentralized. This paper presents a short description of a PVM (Parallel Virtual Machine) toolkit, developed and implemented for the Windows operating system. It is intended to provide a graphical interface for PVM administration and process monitoring. The toolkit will assist the user in the management of the virtual environment, and it can be used for PVM application tuning. The toolkit consists of a remote shell daemon (RSHD), remote shell client (RSH), remote copy client (RCP), and an administrative tool similar to XPVM, which serves administrative and monitoring functions.*

Keywords: distributed environments, PVM environment, distributed processing, administrative tools

1 Introduction

In the past years, computing has been moving from the sequential world to the parallel one, from a centralized organization to decentralized. There is also a large diversity of parallel/distributed computer organizations, including those made possible by high-speed networks. Thus, distributed parallel programming using networked computers became one of the most attractive and cheap ways to increase the computing power [1,4]. In particular, the message-passing paradigm became very popular. PVM (Parallel Virtual Machine) is a software environment, which emulates a distributed-memory multiprocessor in a heterogeneous networked environment. Reasons for the wide acceptance and popularity of PVM for scientific computations include availability of heterogeneous networks and simple programming interfaces. Currently, PVM is available for a variety of hardware and software platforms, from personal computers to supercomputers. Generally, PVM consists of a daemon and PVM libraries, providing fundamental environment functionalities [5]. Additionally, there are a number of auxiliary tools, such as graphical visualization of PVM programs, administrative and other tools, which extend the basic PVM functionally. Unfortunately, such tools are not available for Windows. Therefore, a user must use a text console for all PVM related functions. This text-based interface is hard to use. Additionally, only two programming languages, Visual C++ and Visual Fortran, can be used for programming under PVM for Windows. These are the main motivations for the work presented in this paper. The main objective is to design and implement a PVM Toolkit for Windows, to provide a graphical user interface for PVM administration and process monitoring, and to support multiple languages. The toolkit consists of a remote shell daemon (RSHD), remote shell client (RSH), remote copy client (RCP), and an administrative tool similar to XPVM, which serves administrative and monitoring functions [7]. This paper is organized as follow. Section 2 briefly describes the design. Section 3 presents basic functionalities of RSHD, RSH and RPC. Section 4 presents the

main functionalities of the administrative tool. Finally, section 5 summarizes this work and discusses future extensions planned.

2 Design overview

As mentioned, except for the classical PVM text console, there are no administrative tools supporting the PVM implementation for Windows (WINPVM). Moreover, only two programming languages: Visual C++ and Visual Fortran, can currently be used for programming under PVM for Windows. In this work, we extend the PVM implementation for Windows [2,3]. First, we provide a graphical monitoring console similar to XPVM, to be used for environment management and to assist in tracing and tuning performance of specific applications. Second, we provide additional useful tools, and extend the standard functionalities of other tools:

- A new RSHD server, allowing remote process invocation to UNIX computers (feature not available in the standard PVM for Windows).
- RSH shell, allowing adding new hosts to execute in PVM (there are some commercial version, but very costly).
- RCP client, allowing distributing the application code.

All *pvm* functions placed in the PVM libraries communicate with the user utilizing the *printf* family of functions. It causes some problems with programming under WINAPI. So, to allow easier interfacing in the graphical mode, a new *pvm_print* function, presented below, is provided in the PVM libraries (module `newcrt.c`).

This function checks if the invoked PVM library function is to be used in the graphic mode. If so, an internal Windows Message signals to the output window to display the incoming messages in the graphic mode. This allows very effective use of the PVM libraries in the graphical Windows environment - the libraries do not have to be modified.

```
void pvm_printf(char *fmt, ...)
{ va_list va;
  if (fmt)
    { va_start(va, fmt);
      Vsprintf(commBuff, fmt, va);
      Va_end(va); } /* if */
  #ifdef WINMAIN
  if (applHandle)
     SendMessage(applHandle,
       PRINTFMESSAGE, commBuff, 0);
  else  #endif
  printf("%s", commBuff); }
```

The above tools allow us to design and implement the graphical console. Similarly to XPVM, to capture tracing information at runtime, the user only needs to compile his program using PVM libraries (some additional changes in the standard *tracer* were needed). Data captured during execution is immediately presenting (and animated) by the new graphical console. All tracing information can be also stored in external files for later use such as for post-mortem analysis. Additionally, the user programs can be written in Borland C++ and Borland Delphi in addition to Visual C++ and Fortran.

3 Remote Shell Daemon RSHD

RSHD daemon acts as a server for remote shell (RSH) and remote file copy (RCP). RSHD is a service available on the UNIX operating system. Therefore, we need to provide RSHD for Windows in order for PVM to communicate with UNIX. The new RSHD uses the same algorithm

Fig.1. The main window of the Remote Shell Daemon

as RSHD for UNIX BSD [6]. Additionally, it is equipped with a graphical user interface, which allows changes to the execution environment without having to restart it. The new daemon serves some useful functions such as: allowing debugging, verifying the. `.rhost` file, checking secure connection conditions, manual and remote testing of Windows architecture (95/98, NT, 2000), redirecting `stdout` and `stderr`, choosing the localization of "temp files" ($TEMP or current directory). The RSHD graphical interface is presented in Figure 1.

The main window consists of two parts. The top of window is a menu with the following options: *File|Close* – close the application, *Stop/Start* - close or start the tracing process (the state of RSHD is stored to allow restarting from the last configuration), *Settings* – enable switching on/off the above mentioned functionalities, and *Help* - get the information about the available options.

The bottom of the main window is the central widget where the results of the selected options are displayed. This display can be, at any time, copied to the Windows clipboard. After minimizing, the program is automatically saved as an icon *tray*.

The RSH is a client for RSHD, it is used for executing remote commands on a given host. RSH services include a graphical console, which is divided into four parts. The first part, *Results,* is used for showing the results of execution; the second, *Aliases,* is used for defining the most commonly used operations; the third, *History,* is used for storing the operations; and the last provides the command line. An operation is selected by a click, and subsequently executed by the *Run* button.

The graphical console of the RCP service can work in two modes. First, when the architecture of the chosen host is recognized by the RSH service, the RCP main window is divided into two parts, one for the server and one for the remote host (this allows text copying using the drag and drop method). Second, if the architecture is not recognized, the RCP window provides the standard command line.

4 The PVM Graphical Console for Windows

The PVM graphical console assists the PVM user in management of the virtual environment, and in monitoring the execution of the application. The main window of this console contains three parts, as presented in Figure 2. The top part is a toolbar containing iconic shortcuts for the most often used user commands. The user can customize these. In the illustration, we have: *Quit, Halt, RunRTSK* and *Help. Quit* and *Halt* are equivalent to `quit` and `halt` from the PVM text console*, RunRTSK* starts the tools presented in the previous section, and *Help* presents information about the available options. The com-

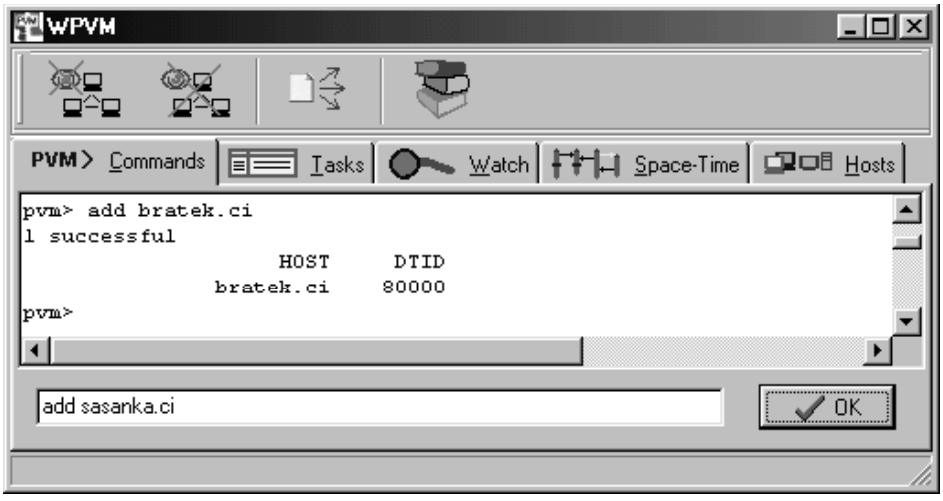

Fig.2. The main window of the administrative PVM console tool

mands are selected with the mouse.

The middle part is the Page Control line containing Tabs Sheets for different visualizations of the virtual machine. This simplifies management of the virtual machine because it allows the user to concentrate on the most desired display features. The available Tab Sheets are: *Commands, Tasks, Watch, Space-Time* and *Hosts*. The default Tab Sheet is *Commands*. The remaining part of the main window is the central widget where the execution traces are displayed, as according to the Tab Sheets. This display can be further subdivided.

This tool can work in two different modes: as a classical text console, or as the graphical user interface for monitoring the virtual machine. Figure 2 illustrates the Tab Sheet *Commands,* which provides the functionality of the classical *text console*, however, with improved interface. The display widget is divided here into two parts: the *result panel*, which presents the results of the recently executed commands in a standard way, and *command line,* where any *pvm* command can be entered. The results displayed in the *result panel* are not lost but instead they are buffered and can be viewed with the scrollbar. They can also be copied in the clipboard. The *command line* can be scrolled using the keyboard arrow keys to retrieve command history. Help is also available under the F1 key. This help provides the available commands when the command line is empty, or the description of the last written command otherwise. The list of commands is also always available under the shortkey *ctrl + F1*.

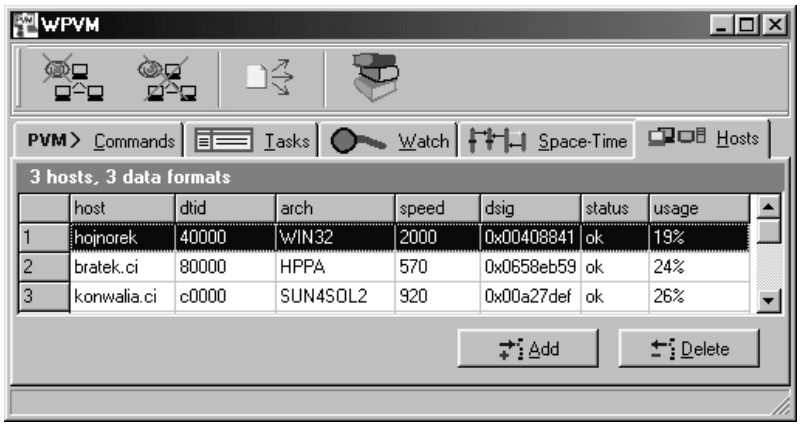

Fig.3. The *Hosts* Tab Sheet window

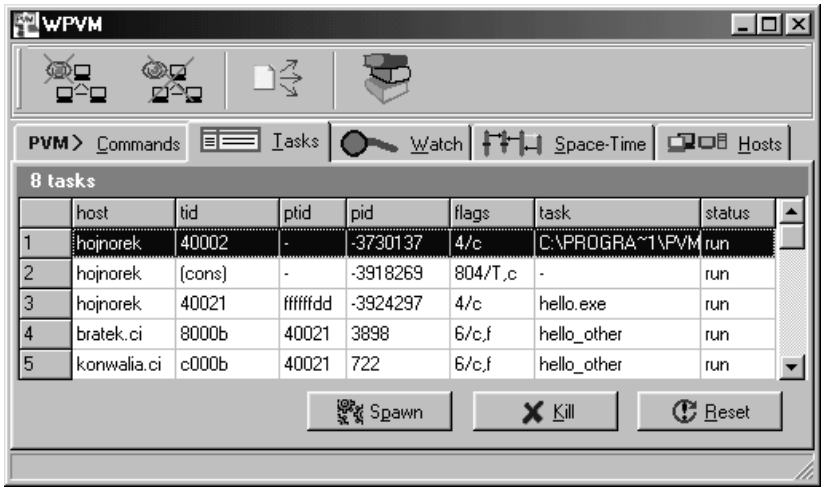

Fig.4. The *Tasks* Tab Sheet window

4.1. Administrative functions of the PVM console

The most often used management functions are commands used for building and reorganizing the virtual machine (*add, delete* and *conf*). These commands are grouped under the Tab Sheet *Hosts,* visible in Figure 3. When this Tab Sheet is selected, the display widget displays the current configuration of the virtual machine. The *add* and *delete* commands are available through two new buttons. A new host can be added either by writing its name in the hosts table, or by choosing it from the list of the recently used hosts, available under the button *add*. A host can be removed from the virtual machine by selecting it in the table and pressing the *delete* button.

Comparing with the information presented by the standard *conf* command, a new column is added providing the usage. As for other PVM consoles, the virtual machine can also be configured using a host file (the name of the file is a parameter of the console start-up), and automatic command realization is also supported through the `.pvmrc` file. When clicking the mouse on the marked host name, the list of running tasks on this host will appear (as illustrated in Fig. 5).

of above example.

The *Task* Tab Sheet, presented in Figure 4, visualizes some general information about all running tasks on the virtual machine at a given moment. Information on a specific task is identified by its *tid – task identificator*. The information is updated without delay (using signals from the *tracer* process). It is also possible to start a new *pvm* process directly from the console (*spawn*), to stop a process (*kill*), and to reboot the machine (remove all of the *pvm* tasks – *reset*). When clicking the *Spawn* button, a new window appears. Using this window, the user can select a process from the process list (directory `$PVM_ROOT\bin\$PVM_ARCH`), define the host architecture for the process, specify other standard *spawn* options, and provide arguments for the process.

4.2. Monitoring functions of the PVM console

A chronological list of all *pvm* functions utilized by the given task (name, time, parameters) is very useful for debugging. Such a list is presented in a separate window and is updated continually (Figure 5). The window can be opened

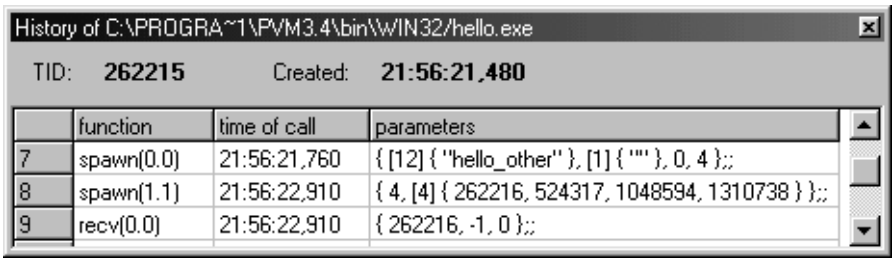

Fig.5. List of functions for a given task window.

The other Tab Sheets are used for tracing running tasks, as well as for visualizing the state of the virtual machine. To demonstrate these, let us consider the following simple example, using a modified version of the standard *hello world* program. The master process generates four slave tasks and waits for answers from them. The slave processes verify that they are spawn, append themselves to the group *hellohellogroup*, synchronize themselves at a barrier, and send confirmation messages to the parent.

The following figures illustrate the execution for any task, and it appears after selecting this task from *Task* Tab Sheet.

Figure 6 shows the *Space-Time* Tab Sheet. It illustrates a window displaying the current state of each task. This window also enables tracing messages sent between the tasks, as visualized on the screen. The lifespan of each task is represented as a vertical line. The color of a line and the corresponding icons indicate the status of the task. The meanings of the icons are summarized in the table 1. The arrows between tasks indicate the parent-child relationship between tasks.

TABLE 1

pvm function	*Icon*	*Function*
pvm_joingroup		attach the process to a group
pvm_recv		waiting for data
Pvm_barrier		waiting on a barrier
Pvm_output		showing of a message

Green color of a task indicates that a task is running. Blue color indicates that a *pvm* function is executing (for example: waiting for data, standing on a barrier, *etc*.). Black lines connecting two tasks indicate a message being between them (*pvm_send*) – it is directed from the sender to the receiver. Each task icon in the figure has some associated parameters, such as the name of the barrier that the process is waiting on, the number of bytes sent, *etc*. The presented graph can be configured for including only specific information (*e.g.*, communication). The number of pixels per second can control the animation speed. It is also possible to choose the tasks to be included in the graph (all, just running, last N tasks, *etc*.).

Figure 6 presents the discussed example in operation. It can be seen that the main process is waiting for data from the spawn slave processes. The slave processes enter the PVM group, are synchronized at the barrier, make some computations, and send the results back to the parent process. It can be observed that even though the slave tasks are identical, their execution time is different depending on the host architecture. Additionally, the window is also equipped with a handy calculator, which can be used by the user.

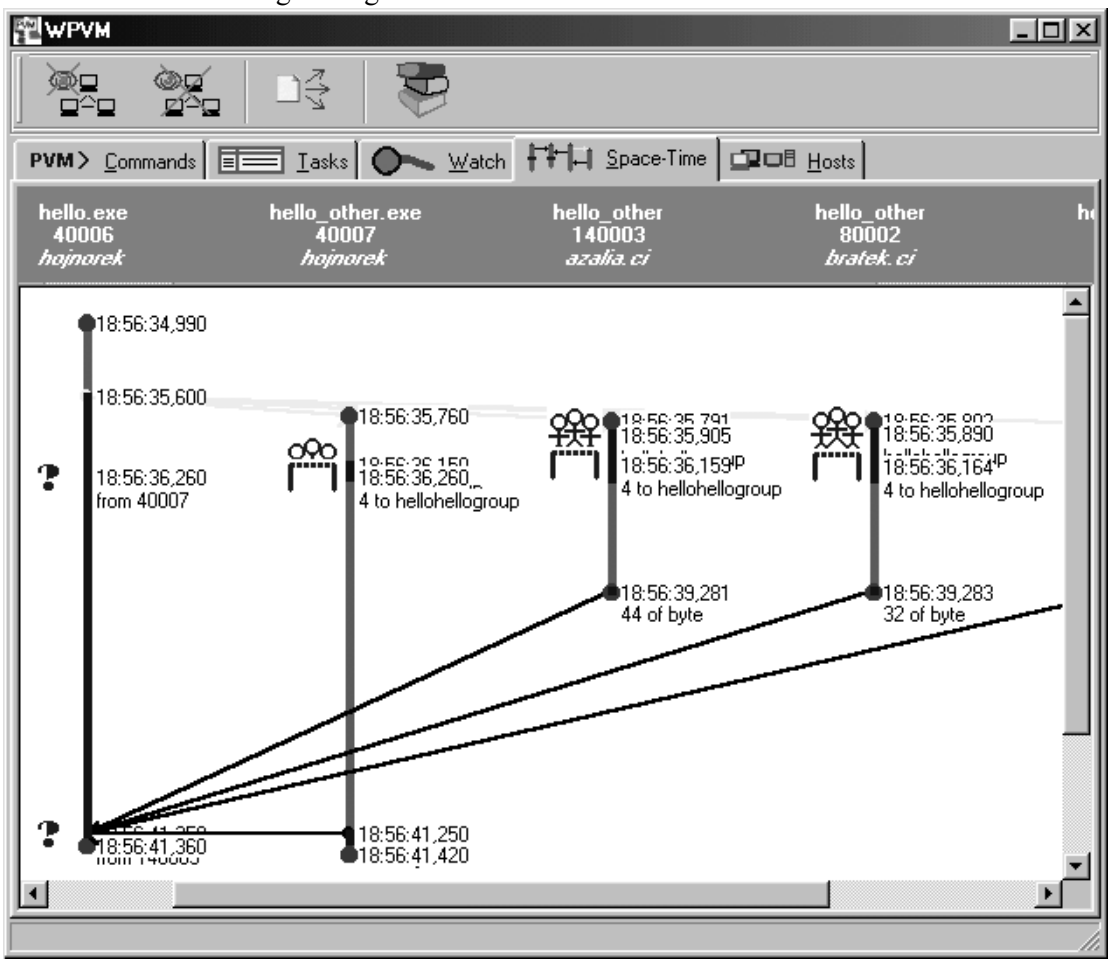

Fig.6. The *Space-Time* Tab Sheet window

Figure 7 shows the *Watch* Tab Sheet (*Network View* under XPVM), which illustrates loads on

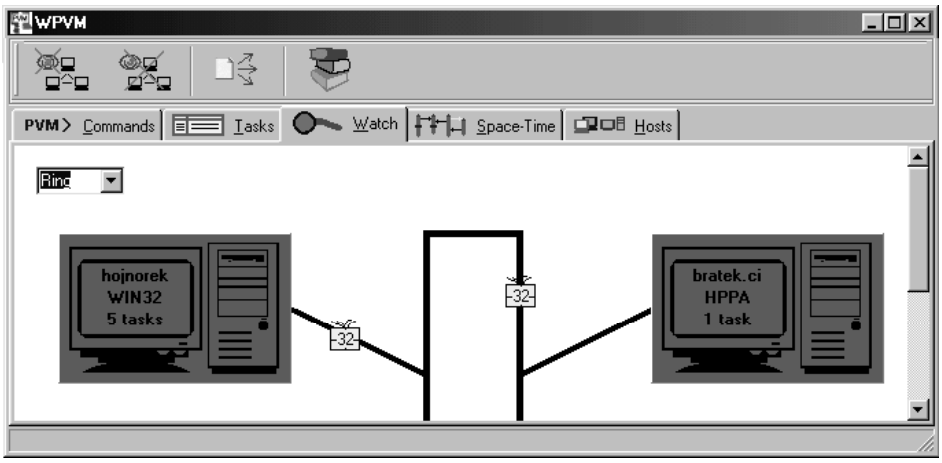

Fig.7. The *Watch* Tab Sheet window

individual hosts and communication in the network. Each host is represented as an icon with its name, architecture, and the number of executing *pvm* tasks. The architecture of the network can also be chosen (seen at top-left window corner) from the following: Linear array, Ring, and Star. The host color indicates if the host is occupied or not (yellow – PVM system task, green – user tasks, blue – no tasks). Clicking on the host icon makes the list of the running tasks available. Additionally, the history of individual *pvm* functions, executed by each task, can also be displayed. The box on connection line symbolizes data sent through the network (it is moving from the sender to the receiver). The number inside of the box indicates the size of the message.

6 Conclusions

This is an ongoing project. At present, a prototype implementation has been completed. Experiments conducted using this prototype indicate that the tool is very useful for the management of the PVM environment under the Windows system, and for performance tuning of PVM applications. In addition, the user can observe temporal graphs indicating the state of each host, the architecture of the network, traffic on the network. Moreover, the toolkit allows two additional languages to be used. The individual tools presented here can be used independently.

References

1. Cosnard M., Trystan D, *Parallel* Algorithms and Architectures, International Thomson Publishing Company, London 1995.
2. Fischer M., Port of PVM to Windows, a port using Myrinet, and a port using Scalable Coherent Interface, available at http://www.markus-fischer.de/projects.htm
3. Fischer M., Announcement of PVM for the WIN32-bit platform, available at www.Paderborn.de/StaffWeb/getin/ntport.htm
4. Foster I., Designing and Building Parallel Programs, Addison-Wesley Pub., 1995 (also available at http://www.mcs.anl.gov/dbpp/text/book.html).
5. PVM: A Users' Guide and Tutorial for Networked Parallel Computing
6. NetBSD Manual Pages, available at http://www.tac.eu.org/cgi-bin/man-cgi
7. XPVM: A Graphical Console and Monitor for PVM, available at http://www.netlib.org/utk/icl/xpvm/xpvm.html