PVM Guide

Paul S. Wang*
Institute for Computational Mathematics
Department of Mathematics and Computer Science
Kent State University, Kent OH 44242
`pwang@mcs.kent.edu`

September 19, 1995

Abstract

The centralized PVM organization and steps users should take to setup their PVM environments are described. Local enhancements to PVM help initialize a new user, compile applications, distribute executables, reduce start-up times for applications, and otherwise manage a pvm. A Common Lisp to PVM interface also has been developed to allow easy inclusion of CL processes in PVM applications. The facilities developed work locally but are of considerable general interest.

1 Introduction

Parallel Virtual Machine (PVM) [1] is a software facility that combines a set of networked computers as a parallel computer. PVM also provides a

*Work reported herein has been supported in part by the National Science Foundation under Grant CCR-9503650

parallel programming environment on actual distributed-memory computers, such as the Cray T3d. In our department, everyone can configure a personal pvm to perform intensive computations or to conduct parallel/distributed computation research. This guide describes our PVM setup and helps a new user get started quickly. The installation configuration and locally developed tools will be presented. Work here extends the original efforts made on the HEAT [4] workstation cluster at Sandia National Labs, Livermore, California.

PVM is installed at a central location available to all users. The PVM root directory is

```
/vol/pvm
```

For every PVM user the shell environment variable `$PVM_ROOT` should be set to this directory. The organization allows sharing of the PVM sources, library, executable code, and tools among all users.

To configure a personal pvm [1], each user should have a main directory to keep per-user application codes and other PVM related work developed by that user. This directory is usually

```
$HOME/pvm3
```

The organization of this directory will be described further in Section 2.

There are also a set of tools that can help the initial setup, the compilation of PVM applications, the distribution of executables to hosts, and the enhancement of performance. These tools are discussed in Section 3.

2 One-time Setup for a New PVM User

There are certain arrangements that a new PVM user must make to set up the environment for a personal pvm. Information here gives the user an idea

[1]We will use *pvm* to refer to a personal parallel machine and *PVM* the software system.

of what must be done. But the procedures need not be carried out explicitly by the user because most of the steps have been automated with tools that will be explained in later sections.

1. Create `$HOME/pvm3` directory.

2. Put the following lines at the end of your `.cshrc` file.

   ```
   ## for pvm
   ## Shared pvm directory for mcs Department
   setenv PVM_ROOT /vol/pvm
   ## directory where new tools are located
   setenv UTIL $PVM_ROOT/UTIL

   if ( ! $?PVMREADY && -f $UTIL/cshrc.stub ) then
       source $UTIL/cshrc.stub
       setenv PVMREADY 1
   endif
   ```

   The `cshrc.stub` is a piece of code common to all users. It adds several directories to your command search path so that your personal applications as well as centralized PVM-specific tools are directly accessible as UNIX commands. It also adds to your man page search path so you can access PVM manual pages on-line. The `cshrc.stub` also defines environment variables for your personal pvm. One is `PVM_ARCH`: the hardware architecture of the local host. Currently there are three architectures `SUN4`, `HPPA`, and `SGI`.

3. Create `$HOME/pvm3/local_disk_bin` and `$HOME/pvm3/bin/$PVM_ARCH` directories. For our installation, `$HOME/pvm3/bin/$PVM_ARCH` is assumed to be on a shared disk accessible from all hosts. The symbolic link

```
$HOME/pvm3/local_disk_bin
```

points to a directory that is on a disk local to a host. For example in `$HOME/pvm3/`, you can execute the following line,

```
ln -s /tmp/pvm_bin.$USER local_disk_bin
```

to specify a local-disk directory (`/tmp/pvm_bin.$USER`). This directory will contain executables for PVM applications that are distributed from `$HOME/pvm3/bin/$PVM_ARCH` at compile time. This scheme is adopted with the assumption that starting PVM applications from local disks is more efficient than starting them from a single shared (NFS-mounted) disk. For example, runtime loading of `$HOME/pvm3/bin/SUN4/a.out` to `tiger`, where `$HOME` is shared among all hosts, is much slower than loading it from `/tmp/pvm_bin.$USER/a.out` local to `tiger`.

4. If you set the command search path in your `.login` file, make sure you do not undo the work done in the `.cshrc`. To make things work, you may need to add the following lines to your `.login` file somewhere below the point where you set the search path.

```
if ( -f $UTIL/cshrc.stub ) then
    source $UTIL/cshrc.stub
    setenv PVMREADY 1
endif
```

You also need to establish a `hostfile`, under `~/pvm3` for example, which contains the host name of each computer you wish to be included as part of your pvm.

3 Useful Tools

There is a set of tools to aid PVM related work and experimentation. These tools are contained in the directory `$PVM_ROOT/UTIL`. Although they have been added for work here in the Department, these tools would be useful on many other PVM installations.

`cshrc.stub`

This is a specialized version of the PVM-supplied file. It is customized to be shared by all users, as their individual `.cshrc` sources this file. The effect of this shell script file is to add PVM related executable directories to a user's command search path. It also sets important environment variables and modifies the man page search path. In addition to this script, there are a number of commands.

Each command is explained briefly below. The following points apply to each command.

- There is an on-line manual page.
- A command given with the wrong number of arguments displays its usage.
- Command options must be given in the specified order.
- If an optional hostfile argument is not given, then the file `hostfile` in the current directory or in the `$HOME/pvm3`, in that order, is used.

3.1 `pvminit`

Instead of doing steps 1 – 3 in Section 2 manually, a new PVM user may invoke the command **pvminit** which performs each of these steps automatically. This is a much quicker and safer approach.

```
### Usage:   pvminit [ -R pvm_root_dir ] [ -L local_bin ]
```

The PVM root directory (value of `$PVM_ROOT`) is specified by the `-R` option or defaults to `/vol/pvm`. The `-L` option gives the directory name on every local disk to house application executables. This value defaults to `/tmp/pvm_bin.$USER`. Normally, you simply issue the command

pvminit

3.2 pvmcc1

The **pvmcc1** command compiles a C-coded PVM application for you on the local host. It generates an executable file in

```
$HOME/pvm3/bin/$PVM_ARCH
```

and distribute this executable to the local disk directory (`pvm3/local_disk_bin`) for the local host. a list of hosts with the same architecture as the host.

```
### Usage:
   pvmcc1 [-U compiler] [-N a.out] [-D hostfile] [-G]
  [-c] file-or-option ...
```

The name of the executable can either be entered interactively or specified with the `-N` option. The compiler used is cc or specified by the `-U` option. If the `-D` option is given then **pvmcc1** will distribute the executable to all the hosts of the same architecture as the local host contained in the given hostfile. Specify the `-c` option as indicated for **pvmcc1** to generate `.o` files only (no executable produced, no distribution made). To compile source files using the PVM Process Group library and create an executable, be sure to include the `-G` option as indicated.

3.3 pvmf77l

The Fortran 77 counterpart of **pvmcc1**. The default compiler used is f77. Some systems, such as HPUX, offer the **fort77** compiler which may be requested by the `-U` option.

3.4 pvmcc

Given a set of C source programs, **pvmcc** compiles them for your entire virtual machine by generating an executable on each host specified in a given hostfile. This is done by compiling once on each different architecture and distributing to all specified hosts.

```
### Usage:  pvmcc [-U cc] [-N a.out]
  [-H hostfile] args_for_pvmcc1
```

The name of the executable is either entered interactively or specified with the `-N` option. If the `-H` option is not specified, the file `hostfile` in the current working directory is used.

A file argument given to **pvmcc** must be such that it remains valid when used on another host in your PVM. If **pvmcc** is used from a working directory under your home directory and the files are given relative to the working directory, they will be fine. This is because a user's home directory is unique and reachable via the `$HOME` variable on any host in our Department. Typically, a user uses **pvmcc** in `$Home/pvm3` or a subdirectory.

For example

cd `~/pvm3` **pvmcc** `-N slave slave.c`

will compile `slave.c`, put the excutable `slave` in `~/pvm3/bin/$PVM_ARCH/`, and distribute the correct executable to the local disk for each host in the hostfile.

3.5 pvmf77

The Fortran 77 counterpart of **pvmcc**. The default compiler used is f77.

3.6 pvm_distrib

Given a set of PVM executable programs, **pvm_distrib** distributes them to the local disks for the hosts indicated in the given hostfile. The hostfile may contain hosts of many different architecture types, but **pvm_distrib** will distribute them only to those of the same architecture type as the machine the command is being run from, or a specific architecture given by the `-T` option.

```
### Usage:
pvm_distrib $0 [-T architecture] [-H hostfile] a.out1 ...
```

3.7 pvmexec

This command executes the given command string on each host specified in the hostfile. The actions are carried out in parallel and all output, including error output, is sent to the file `/tmp/$HOST.log` on each local host.

```
### Usage: pvmexec [-H hostfile ] command-string
```

For example, the command

pvmexec `'/bin/rm /tmp/pvm_bin.$USER/a.out'`

will rid you of all the `a.out` files in the `/tmp/pvm_bin.$USER` directory on each host listed in the given `hostfile`.

3.8 `pvmclean`

This simple command helps a user clean up the local disk directories

`$HOME/pvm3/local_disk_bin/`

Simply do

pvmclean [hostfile]

To clean those directories on the given hosts.

4 Creating and Running PVM Applications

You write a program that uses multiple processes to perform tasks in parallel on a pvm. Each process can be written in C, f77, or Common Lisp. The hosts forming a pvm is given in a hostfile. Hostfiles can be configured to suit individual applications.

Follow these steps to run an application on your personal pvm:

(i) Create a hostfile to define a configuration of machines.

(ii) Generate executables for each architecture in the configuration.

(iii) Distribute the executables to each host in the configuration. (This step is actually optional—the executables could be put in a single shared location—but should improve startup efficiency.)

(iv) Start personal pvm.

(v) Run the application on any host in your personal pvm.

(vi) Halt pvm.

Here is a recommended sequence to achieve the above steps:

(a) Use

pvmcc1 `-c` *file1*`.c` *file2*`.c` ...

to compile your program and fix any problems found.

(b) List the hosts to form your pvm in a file (`pvm3/hostfile`)— one host name per line, comment lines start with `#` (an enhancement not available in the PVM distribution). Then run **pvmcc** to distribute the executable to all hosts. For example,

pvmcc `-H` *my_hostfile* *file1*`.c` *file2*`.c` ...

(c) Run

pvmd3 *hostfile* `&`

to start your parallel virtual machine.

(d) Now you are ready to run your application on your pvm. Simply run your executable main program from one of the hosts:

a.out

Usually the main program starts other processes on other hosts to exploit parallelism.

(e) After you are done you can choose to leave your personal pvm running to be used again later, or you can take it down. To shut your pvm down, issue the command

pvm (pvm console)

then type halt at the `pvm>` prompt.

```
pvm> halt
```

To exit the **pvm** command without halting your personal pvm do

```
pvm> quit
```

The **pvm** command runs the *pvm console* and can be issued on any host within your pvm. It is an interactive control program that allows you to add, delete hosts, spawn tasks, display information on tasks, etc. Issue `help` at the `pvm>` prompt to see a list of the available commands. The pvm console also reads the init file `$HOME/.pvmrc` which should contain console commands. Do **man** `pvm` for more details.

5 Required Header Files

In you application program source files make sure you use the line

```
#include "pvm3.h"
```

in C files and the line

```
      include '/vol/pvm/include/fpvm3.h'
```

in Fortran files.

6 Using Makefiles with PVM

PVM provides a utility `aimk` which uses a `Makefile.aimk` in the current directory to help build PVM related files.

If you wish to use `aimk`, the easiest way is to copy the example file to your own directory:

```
cp $PVM_ROOT/examples/Makefile.aimk $HOME/pvm3
```

This will allow you to copy the examples to `$HOME/pvm3` and compile them using `aimk`. You should edit `Makefile.aimk` to add your own applications. You may also choose to use **pvmcc1**, **pvmcc**, or **pvm_distrib**, in the `Makefile.aimk` for better effect.

7 Documentation

On-line manual pages for PVM, including those for the tools described in this guide, are accessible using the UNIX `man` command on any machine where man pages work. This can be done by including `$PVM_ROOT/man/` as part of the `MANPATH`. Normally `MANPATH` is set for you automatically in `cshrc.stub`. In case it is not, you can modify `MANPATH` yourself.

Other documentation that comes with the PVM distribution is kept in `$PVM_ROOT/doc/`. A LaTeX version of this report, and on-line man pages are in `$PVM_ROOT/UTIL/`.

8 Performance Considerations

A key improvement of the PVM setup described here over the regular PVM configuration is the use of the symbolic link

```
$HOME/pvm3/local_disk_bin
```

This feature deserves some discussion.

Usually, PVM is used in an environment where a group of UNIX workstations and other high speed servers are connected by a fast LAN within a single department. Each PVM user requires **rsh** privilege on each component of the parallel virtual machine. Thus, the computers are usually under the administrative control of one closely knit group. Often, the users will keep unique home directories on centralized disk servers to avoid file duplication and to simplify system maintenance and file backup. Furthermore, executable codes common to all, or a subset, of the machines are also centralized for the same reasons.

Our PVM setup and tools take these conditions into account. All PVM sources, libraries, shell scripts, etc. are kept on shared disks. But, the PVM executables for each individual computer can be kept on local disks to minimize time required to spawn PVM processes and to avoid service congestion on shared disks. Because loading an executable from a locally mounted disk can be many times faster than fetching it via NFS (especially during multiple parallel fetchings) , this reduces the `pvm_spawn` overhead significantly. For certain applications where spawning of medium grain activities are performed repeatedly this can be very important.

9 A Common Lisp Interface to PVM

There is also a CL-PVM interface which makes running programs written in Common Lisp (CL) as PVM tasks possible. The CL-PVM interface, developed at ICM/Kent, provides one CL routine to each PVM call. For example, the CL function `pvm-spawn` calls the C function `pvm_spawn`.

Here is a sample call in Lisp

```
(pvm-spawn "hello_other" "" 0 "nimitz.mcs.kent.edu" 1 'psids)
```

The interface code is `clpvm3.o` compiled by the CL compiler (**lc**) from the source file `clpvm3.lsp`. In addition the code `clgpvm3.o` interfaces to the PVM Process Group library. We use AKCL in this department and the command **akcl** invokes the lisp system.

A lisp process can be turned into a pvm task by writing a simple shell script. Consider the lisp program `slave.lsp` with a top-level call `(slave)`. The following *csh* script can be used:

```
### slave.lsp.csh  a script for the lisp slave task

/usr/local/bin/akcl <<EOF
;; load cl interface to PVM
(si::faslink "$PVM_ROOT/UTIL/clpvm/lib/$PVM_ARCH/clpvm3.o"
     "$PVM_ROOT/lib/$PVM_ARCH/libpvm3.a -lc")
;; load lisp program
(load "$HOME/pvm3/slave.lsp")
;; or (load "$HOME/pvm3/bin/$PVM_ARCH/slave.o")
;; or (load "$HOME/pvm3/local_disk_bin/slave.o")
(slave)              ;; execute top-level lisp routine
(bye)                ;; lisp process exits
EOF
```

Add similar code to load `clgpvm3.o` after `clpvm3.o` if needed. Note, the first two steps in this shell script can be time consuming. It is possible to pre-load them into a spcialized lisp image to make invocation much faster.

Using a shell script to invoke a lisp process has other advantages: the program is architecture independent. Thus, you can simply distribute the program to all suitable hosts with the **pvm_distrib** tool. The commands **pvmlc** and **pvmlc1** help compile and distribute lisp object files to local disks. A script can can also help testing and debugging as show by the following code:

```
#!/bin/csh
###debuging lines commented out
###set logfile = /tmp/${USER}LOG.$$
###echo $0 INVOKED >! $logfile
###/usr/local/bin/akcl >>&! $logfile <<EOF

/usr/local/bin/akcl > /dev/null <<EOF
```

```
(si::faslink "$PVM_ROOT/UTIL/clpvm/lib/$PVM_ARCH/clpvm3.o"
     "$PVM_ROOT/lib/$PVM_ARCH/libpvm3.a -lc")
(load "$HOME/pvm3/local_disk_bin/hello_other.lsp")
(hello_other)
(bye)
EOF

###echo FINISHED >>! $logfile
```

The **pvmcltask** command can be used to generate a pvm executable task in the form of a *csh* script when given a list of lisp object files. See the on-line man page for details. The generated script can be edited and later distributed by **pvm_distrib**.

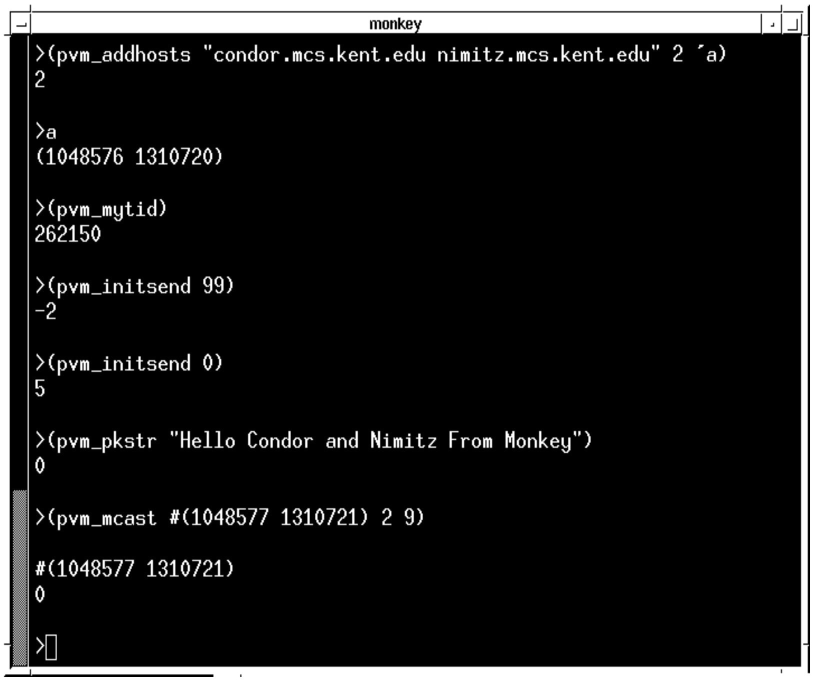
With a Lisp process, one can also run PVM calls interactively as shown in Figure 1. See [3] for more details.

10 Maintaining and Updating PVM

All of the tools added to the PVM distribution have been kept under the directory `$PVM_ROOT/UTIL`. Thus, after installing a PVM update simply `mv` this directory in place.

There is one other thing to do before compiling a new PVM distribution. Modify `pvm3/src/global.h` to add the correct executable search directory. A copy of the file is in `$PVM_ROOT/UTIL/add/`. Basically, the following modification is made

```
/* modified to use local disk for executables
#define DEFBINDIR "pvm3/bin/$PVM_ARCH:$PVM_ROOT/bin/$PVM_ARCH"
*/
```



```
>(pvm_addhosts "condor.mcs.kent.edu nimitz.mcs.kent.edu" 2 'a)
2

>a
(1048576 1310720)

>(pvm_mytid)
262150

>(pvm_initsend 99)
-2

>(pvm_initsend 0)
5

>(pvm_pkstr "Hello Condor and Nimitz From Monkey")
0

>(pvm_mcast #(1048577 1310721) 2 9)

#(1048577 1310721)
0

>
```

Figure 1: Interactive Use of PVM from Lisp

```
#define DEFBINDIR "pvm3/local_disk_bin:pvm3/bin/$PVM_ARCH:\
                   $PVM_ROOT/bin/$PVM_ARCH"
#endif
```

References

[1] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM 3 User's Guide and Reference Manual.* Technical Report ORNL/TM-12187, Oak Ridge National Labs, Oak Ridge, TN, May 1993.

[2] Michael J. Lewis and Raymond E. Cline, Jr. "PVM Communication Performance in Switched FDDI Heterogeneous Distributed Computing Environments," *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, pp. 13-19, October 1993.

[3] Liwei Li and Paul S. Wang, "CL-PVM: A Common Lisp Interface to PVM," Proceedings[2], PVM User's Meeting, Pittsburgh, May 7-10, 1995. (Also ICM Technical Report ICM-199509-01, `http://www.mcs.kent.edu/cgi-bin/publish/access`)

[4] Paul S. Wnag, Michael J. Lewis, and Raymond E. Cline, Jr. "PVM Installation, Usage, and Performance on HEAT", Internal Technical Report, Sandia National Labs, Livermore, Calif.

[2]`http://www.cs.cmu.edu/Web/Groups/pvmug95.html`