Finalization report: homogeneous PVM/PARIX

Overeinder, B.J.; Sloot, P.M.A.; Petersen, J.

Publication date
1994

Commission of the European Communities

****

ESPRIT III

PROJECT NB 6756

****

CAMAS

COMPUTER AIDED MIGRATION OF APPLICATIONS SYSTEM

****

CAMAS-TR-2.3.4
Finalization Report
Homogeneous PVM/PARIX

****

Date: October 1994

ACE - U. of Amsterdam - ESI SA - ESI GmbH - FEGS - Parsytec GmbH -
U. of Southampton.

Technical Report CAMAS-TR-2.3.4

Finalization Report
Homogeneous PVM/PARIX

B. J. Overeinder and P. M. A. Sloot
University of Amsterdam

J. Petersen
Parsytec GmbH

October 1994

Abstract

This document reports on the design and implementation considerations of PVM/PARIX, homogeneous version 1.0. This version is for use with PARIX 1.2 only. Further, it contains information how to use Homogeneous PVM/PARIX and the appendix contains the installation notes.

1 What Is Homogeneous PVM/PARIX?

The popularity of PVM nowadays, can be partly explained by the inherent portability of PVM programs over a large number of parallel systems. The spectrum of parallel systems consists from loosely coupled networks with remote shell capabilities, via parallel clusters, to MPP architectures.

The use of PVM as a parallel programming environment for MPP architectures, raises the following observation. The heterogeneity which made PVM so popular in the beginning, now seems a drawback in the application of PVM to MPP architectures due to the large overhead to handle this heterogeneity. This is especially true for the communication primitives, e.g., communication latencies between processes inside the MPP can become quite large. This has motivated the development of a so-called homogeneous version of PVM for the PARIX parallel operating system.

This PVM/PARIX version has the additive homogeneous version, which essentially means that all PVM processes are to be run on the MPP system only. Thus, the parallel virtual machine can only consist of the nodes in the Parsytec MPP, and PVM inter-process communication with the front-end system is not supported (of course I/O with the front-end is possible, this is handled by PARIX). It is therefore not possible to send messages to PVM tasks outside the MPP.

The large advantage however, is that the communication latency and throughput will be improved drastically, resulting in a communication performance which is almost as fast as the underlying PARIX layer.

The user interface of this Homogeneous PVM/PARIX version is somehow different than with the standard PVM. There is no console present, for example to start a virtual machine, and the Homogenous PVM/PARIX programs are issued as normal PARIX programs, e.g., with the `run` or `hpvmrun` script (see also Section 4).

The homogeneous version of PVM for PARIX, is an implementation of PVM 3.2.6 [1] on top of PARIX 1.2 [2]. All PVM functionality applicable to MPP systems has been incorporated, with the exception of `pvm_recvf()`.

2 Design Considerations

2.1 General Overview of Design

The Homogeneous PVM/PARIX version is designed and implemented on top of an asynchronous communication layer, called the Communication Kernel. The Communication Kernel on his turn is implemented on top of PARIX, the operating system for Parsytec's parallel architectures. See also Fig. 1.

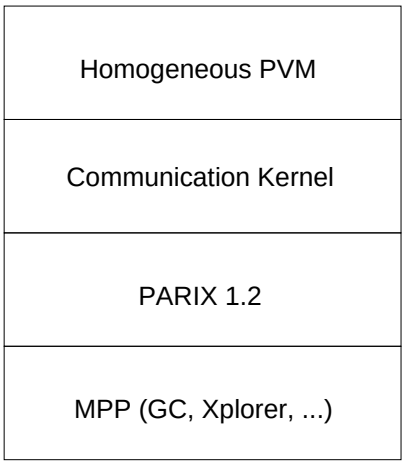

Figure 1: The Homogeneous PVM/PARIX design overview.

The advantage of this layered design is, that the functionality gap between PVM and PARIX is bridged by the Communication Kernel. The used strategy results in an implementation where the PVM intrinsics are clearly separated from the MPP's operating system dependent characteristics. This improves the maintainability and portability to newer PVM versions.

2.2 Design Considerations of the Communication Kernel

The Communication Kernel's primary raison d'être was the need for typed asynchronous buffered communication, which is PVM's basic message passing model. The implementation of the typed asynchronous buffered communication in a separate layer is motivated by the generic characteristics of this message passing model.

Along the design and implementation, other functionality needed for a PVM implementation was thought to be generic enough to be added to the Communication Kernel. As an extension to the point-to-point communication, a multicast communication primitive has been integrated in the Communication Kernel.

Other functionality added to the Communication Kernel is dynamic remote context creation and a run server capable of load balancing according to the round-robin strategy. This two components together efficiently supports the `pvm_spawn` call.

2.3 Design Considerations of Homogeneous PVM/PARIX

With the effective support of the Communication Kernel, the PVM implementation has become quite straightforward. Many of the PVM/PARIX calls are implemented without complications on top of the Communication Kernel. Most prominent implementation efforts to PVM/PARIX were the multiple message buffering scheme and the group server.

The implementation of the PVM multiple message buffering scheme is as flexible as can be. There is no restriction on the number of message buffers, and for each buffer there does not exist a limit on the size.

Group communication and synchronization (`pvm_bcast` respectively `pvm_barrier`) is administered and coordinated by the group server. The group server is a independent thread/context running on processor number 0, but could be placed on any other processor (processor number 0 is always present).

In standard PVM, the group server is build on top of the PVM routines. In our design, the group server is implemented on top of the Communication Kernel, thus resides in the same layer as PVM. This not only improves performance by circumventing an extra layer, but has also the advantage that the group server can make direct use of the multicast provided by the Communication Kernel.

Worth to mention is, that the Homogeneous PVM/PARIX implementation supports multiple (dynamic) PVM tasks per MPP node. Many other MPP specific PVM implementations only supports one task per node.

3 Programming with PVM/PARIX

The Homogeneous PVM/PARIX version is software compatible with other implementations of PVM 3.2.6 as described in [1], with the exceptions discussed in Section 5.

Common Programming Practices

With the Homogeneous PVM/PARIX version, PVM programs become normal PARIX executables, just like any PARIX program you would write yourself. However, the programming practice of writing PVM programs has not changed.

Each PVM program has to include `pvm3.h` (C) or `fpvm3.h` (FORTRAN) found in the `include` directory of the distribution. The first PVM call in any program has to be `pvm_mytid()` (or `pvmfmytid` in FORTRAN programs). This routines initializes the PVM layer and enrolls the task to PVM.

In order to terminate the PVM program, `pvm_exit()` **has to be** called. Without calling this function, each node will hang forever, waiting for the other nodes to call `pvm_exit()`.

Compiling PVM Programs

Compiling PVM programs is like compiling PARIX programs, with the exception of the library that is to be linked with the application. Depending on whether you are compiling a program for

Transputer or PowerPC systems, you prefix the compiler call with `px` for Transputer architectures, and `ppx` for PowerPC architectures. The command-line to compile a program for a Transputer system would look like:

```
$ px cc.px -Ipvmdir/include file.c -o file -Lpvmdir/lib -lpvm3,
```

or using FORTRAN

```
$ px f77.px -Ipvmdir/include file.f -o file \
            -Lpvmdir/lib -lfpvm3 -lpvm3
```

When you want to compile a program that came with a `Makefile`, the simplest way to build the executable is:

```
$ aimk <target>
```

Group Communication Library

The group communication code has been integrated into the standard PVM library. It is therefore not necessary to link with a separate group library in contrast to the standard PVM implementation.

To prevent "old" makefiles from generating all sorts of errors because of a missing group communication library, PVM/PARIX comes with a dummy group library that contains nothing, but keeps the linker happy.

4 Running Programs with PVM/PARIX

Starting Programs

Since PVM/PARIX programs are regular PARIX programs, they can be run using the standard PARIX `run` utility. However, since many PVM makefiles automatically install executables in `$HOME/pvm3/bin/<ARCH>`, using `run` may be inconvenient. Therefore, PVM/PARIX comes with a simple front-end to `run`, called `hpvmrun`, that looks in different places for the binary to execute. The flags and arguments needed to run PARIX jobs should still be given to either `run` or `hpvmrun` (note: `hpvmrun` accepts the same arguments as `run`).

Executing a PVM/PARIX job on a four processors partition becomes

```
$ hpvmrun -a p4 file
```

where `nrm` allocates the four nodes; or with a pre-allocated partition

```
$ hpvmrun -f0 2 2 file
```

Since there exist a number of different `run` scripts, it is impossible to give detailed information on how to run PARIX jobs on your particular system. Please refer to `run(1)` for more information.

NOTE: Do not forget to increase the maximum number of virtual links, if one of the following errors is reported: `_PX_AllocLLink error` or `_PX_AllocVLink error` for PowerPC; and `AllocLLink error` or `AllocVLink error` for Transputer. See also the "Release Notes PARIX 1.2-PPC", Section "Frequently Asked Questions".

PVM Console

There is no `pvmconsole` for the Homogeneous PVM/PARIX implementation, since all tasks can be managed from the command-line. As a result, the output of each of the parallel tasks is sent to the terminal where the `run` or `hpvmrun` was issued.

5 Notes on the Implementation

Unimplemented Calls
This PVM implementation adheres to the definitions in the PVM 3.2.6 manual, with a few exceptions (see also the manual pages [3]). Most notably, the functionality that is not applicable to a homogeneous MPP implementation is not supported (e.g., `pvm_addhost`). Apart from this, several functions not suitable for MPP systems in general are not implemented, such as the signal capabilities. This affects the functions in Table 1.

`pvm_addhost`	`pvm_delhost`	`pvm_kill`
`pvm_notify`	`pvm_recvf+`	`pvm_sendsig`
`pvm_start_pvmd`	`pvm_tickle`	

Table 1: The functions indicated with a '+' will be implemented in a future release of PVM/PARIX, the other functions are not applicable to a MPP system and will therefore not be implemented.

All non-implemented functions return `PvmNotImpl`.

Process Grids
Since PVM/PARIX is built on top of PARIX, it has also the same limitations with respect to the grid that can be built, which implies that one can only run PVM programs on a $m \times n$-grid.

Different Libraries
The Homogeneous PVM/PARIX distribution comes with a standard PVM3 library, and one compiled with debug information (`libpvm3g.a`). The last might be useful when you encounter a bug in PVM/PARIX and want to fill in a bug-report form.

6 Performance Results

A ping-pong experiment was performed to measure the communication performance of the Homogeneous PVM implementation. The typical ping-pong experiment used on Parsytec parallel architectures is the SendLink/RecvLink benchmark, which sends packages of size 1, 4, 16, 64, 256, 1K, 4K, 16K, and 64K bytes. Each measurement for a package size is repeated thirty-two times, resulting in a mean value.

The time measurements in the ping-pong experiment are performed in the following way:

```
t1 = time();
send();
receive();
t2 = time();
elapsed_time = (t2 - t1) / 2;
```

The latency measured in Table 2, is the time to send and receive one byte to and from the nearest neighbor in the process grid. The throughput is measured by sending a message of 64K bytes to and from, and divide the number of bytes by the elapsed time.

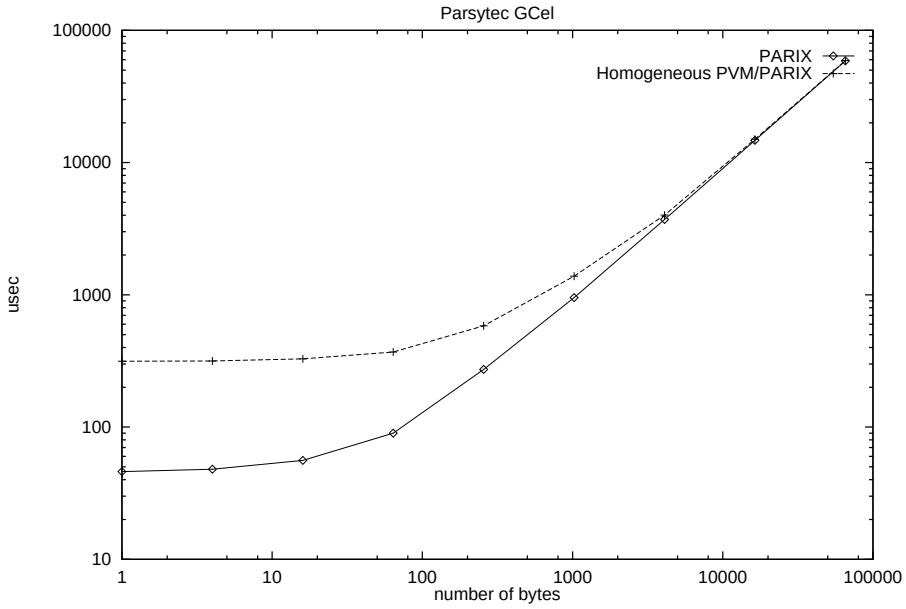

Figure 2: Communication performance of PVM versus PARIX on Parsytec GCel.

Machine/OS	Latency μsec	Throughput (Kb/sec)
GCel/PARIX	46	1088
PVM	315	1082
Xplorer/PARIX	89	1032
PVM	152	1027

Table 2: Communication latency and throughput

A Installing Homogeneous PVM/PARIX

Where to Put PVM

PVM can be installed either in a users home-directory, or in a system-wide accessible directory to ease sharing between different users. In any case, there must be a directory named `pvm3` somewhere on the filesystem. The complete path to this directory, including `pvm3` will be denoted by *pvmdir* in this manual.

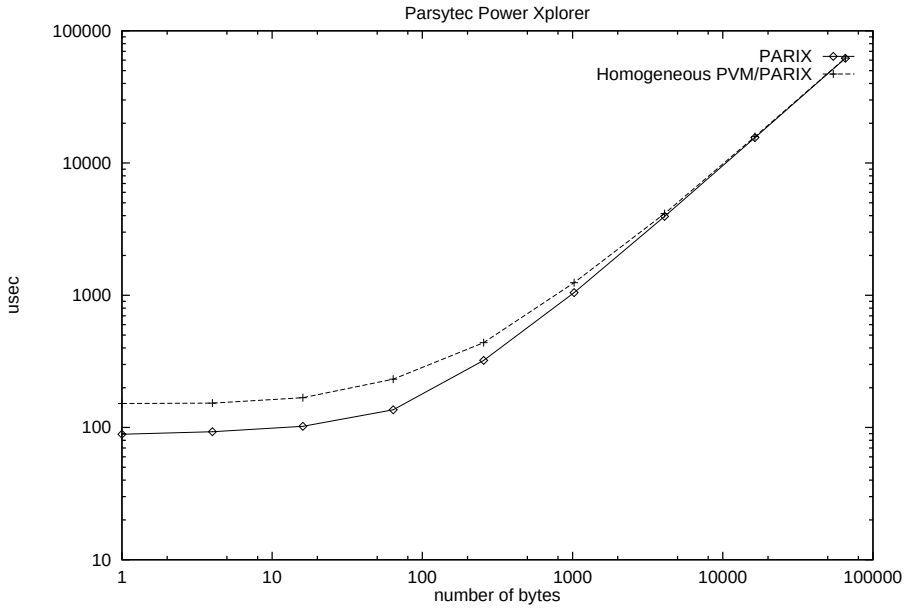

Figure 3: Communication performance of PVM versus PARIX on Parsytec Power Xplorer.

Apart from this distribution tree, each PVM user has to create a `pvm3` directory in his home directory. This directory will contain his own PVM-executables.

The Distribution
The Homogeneous PVM/PARIX distribution contains two files:

- `install`;
- `HPvmParix-1.0.tar.Z`.

The tar-file is organized as follows:

`pvm3/bin`	Directory containing several example programs
`pvm3/doc`	Some documentation on this PVM implementation
`pvm3/examples`	Sources of standard PVM example programs
`pvm3/include`	Include files for C and FORTRAN programs
`pvm3/lib`	Libraries for PVM/PARIX, together with supporting tools
`pvm3/man`	Manual pages for Homogeneous PVM/PARIX

One megabyte of disk space is required for the complete Homogeneous PVM/PARIX installation.

The Installation
First, place the install script and the Homogeneous PVM/PARIX distribution somewhere on your

system. The install script with its arguments takes care for the installation in the proper directory. To actually install the Homogeneous PVM/PARIX distribution, execute the install script:

```
$ ./install <pvmdir-path>
```

where <pvmdir-path> is the path where the *pvmdir* should be created. For example, if you want PVM to be installed in /usr/local, you should run

```
$ ./install /usr/local
```

which creates /usr/local/pvm3 and unpacks the distribution into it.

If a different PVM version is already present, the install script will rename some of the existing tools in order to retain compatibility with the other version. You will not notice any difference when using either PVM version: every change is made completely transparent.

NOTE: If you plan to (re-)install a PVM version other than the heterogeneous PVM/PARIX in the same directory where Homogeneous PVM/PARIX resides, take care of the following.

- Before (re-)installation, make sure that you save the files pvmgetarch and aimk from *pvmdir*/lib before you install the new version.
- After (re-)installation of the new PVM version, rename the files pvmgetarch and aimk that come with the new distribution to pvmgetarch.org and aimk.org.
- Finally, restore the saved (Homogeneous PVM/PARIX) version of these programs.

This is necessary because the Homogeneous PVM/PARIX version of these tools are *wrappers* around the original tools and thus rely on the originals being present, renamed to <name>.org

Per User Installation

In order for a user to use PVM, he must create a pvm3 directory in his home directory to contain his own PVM-binaries. To create this directory, execute the following commands:

```
$ mkdir $HOME/pvm3
$ mkdir $HOME/pvm3/bin
$ mkdir $HOME/pvm3/bin/<ARCHITECTURE>
```

where <ARCHITECTURE> is either PARIXPPC_H for the Homogeneous PVM/PARIX for Power PC systems, or PARIXT8_H for the Homogeneous PVM/PARIX for Transputer T800 systems.

Last, but not least, the path containing the PVM tools (*pvmdir*/lib) should be added to the user's PATH (command path) environment variable and the environment variable PVM_ROOT should be set to *pvmdir*.

Different PVM Versions

As with standard PVM, pvmgetarch is supplied to determine on what system PVM/PARIX is running. This script is a wrapper around the original pvmgetarch, which, if present, is renamed to pvmgetarch.org for this version.

If a particular computer system (front-end) has shared file systems with other front-end computers, the problem can arise that it is impossible to determine which PVM version has to be chosen.

Should it be Homogeneous PVM/PARIX on a PPC based system, Homogeneous PVM/PARIX on a T800 based system, or the heterogeneous version?

To solve this problem, you can set the environment variable `PVMPARIX` to `PARIXPPC_H`, `PARIXT8_H` or to `NO`. If `PVMPARIX` is set to `PARIXPPC_H` or `PARIXT8_H`, the `pvmgetarch` script determines the appropriate PVM version for PPC systems or T800 systems. If `PVMPARIX` equals `NO`, the control is immediately given to the original `pvmgetarch` script to determine which PVM version you need.

References

[1] Al Geist et al. PVM 3 user's guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee, May 1993.

[2] Parsytec GmbH. *PARIX 1.2 Documentation*.

[3] University of Amsterdam and Parsytec GmbH. *Homogeneous PVM/PARIX Manual Pages*, October 1994.