Commission of the European Communities

****

ESPRIT III

PROJECT NB 6756

****

CAMAS

COMPUTER AIDED MIGRATION OF APPLICATIONS SYSTEM

****

CAMAS-TR-2.3.3
PVM/PARIX Progress Report

****

Date: March 1994

Review 3.0

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

Technical Report CAMAS-TR-2.3.3

PVM/PARIX Progress Report

F. vd Linden, J. Petersen and P.M.A. Sloot
University of Amsterdam

March 1994

1 Outline

This document describes the heterogeneous version of PVM/Parix, an implementation of PVM for Parix, the runtime system for Parsytec computer systems. This work was done in accordance with task 2.3 of the CAMAS project.

2 Introduction to PVM

PVM (which stands for Parallel Virtual Machine) is a parallel programming environment. It provides a generic parallel programming interface, and a runtime system.

The system runs on a heterogeneous collection of machines, connected by a local network. These machines can include different kinds of workstations and/or parallel machines.

PVM is not restricted to a single parallel programming paradigm, because of the flexible implementation of the runtime system; hybrid schemes can be used. This makes it possible to implement (parts of) a problem using the optimal model.

The heterogenity and support for multiple parallel programming models, make PVM a system that can easily be used to implement portable parallel software on a variety of systems.

3 Design description

The heterogeneous version of PVM for Parix was designed to be totally compatible to the existing network-PVM implementation, yet to be efficient when working just on a Parix system. Below are some issues that had to be dealt with.

- PVM wants point to point communication, without knowing about any topology. It does not provide its own routing. It's send operation is non-blocking, the receive can be done both blocking and non-blocking.

  To be fast in communication, the choice was made to use virtual links. Virtual links are the fast and reliable way to communicate on Parix.

- Virtual links provide blocking communication, and that is not always what PVM wants, as noted above. To provide non-blocking communication, threads have to be used to handle the communication, while the main code keeps on executing. Since the failure of one node to receive a message should not lead to a stop in communication from the node wanting to send to it, the choice was made to provide a thread for each virtual link.

- PVM wants to be able to send/receive directly to any task. PVM provides no routing itself, and doing the routing on top of Parix, which already has routing in it, is certainly not optimal. Using a totally connected network of nodes (via virtual links) was the one solution that solved this problem. One disadvantage of this approach is, that the large amount of links needed takes up a lot of memory when using large numbers of nodes.

In the standard code of PVM version 3, some basic support for multiprocessor systems was already available. It was based on asynchronous communication, and it implemented the necessary buffering that provided compatability with messages from other host types. These messages may arrive via, for example, a local network, and are broken into pieces by the network-PVM code. This message disassembling and reassembling is quite an overhead, but unavoidable in the heterogeneous implementation.

The asynchronous communication wanted by the standard PVM 3.0 code was not readily available in the Parix libraries (it needed, for example to know on a per-message basis whether communication was done). This is why an extra communication layer was added. This layer also provides the necessary transparent communication from nodes to the front-end machine (the front-end is seen as another node by the layer).

The library code of PVM 3 uses some global internal buffer structures, and is not suitable for multi-thread usage. The heterogeneous version of PVM does not provide explicit placing of tasks on multiprocessor nodes, so the possibility of using multiple PVM tasks on one node did not seem very advantageous. This is why no support is provided for multiple PVM tasks on one node in the heterogeneous version.

The standard code of PVM 3 handles a multiprocessor machine by running a special version of the PVM daemon on the front-end. This daemon handles all requests for the multiprocessor. The rest of the PVM system only sees one machine through this daemon: the multiprocessor. The daemon wants to be able to handle requests, and put tasks on each individual node on the multiprocessor machine. In Parix, executing code from on a specific node is initiated from the node itself, by using the Execute() call. To mimic an execute that is initiated from the front-end (the daemon, to be exact), a small server has to run on each node, listening for requests from the daemon.

4 Status

The heterogeneous version 1.0 has been released through Genias Software Gmbh, and is also available to the CAMAS consortium.

The following appendices describe respectively: ‘Introduction to PVM/Parix’ and PVM/Parix design and implementation description’.

Appendix I Introduction to PVM/PARIX

Overview

Parallel Virtual Machine (PVM) has become the most popular and widely used parallel programming environment. Over 10,000 users have retrieved PVM from various FTP sites. PVM is available now on more than 40 architectures including workstation networks, vector computers, and shared and distributed memory parallel computers. An uncounted number of scientific and engineering applications rely on PVM as a portable parallel programming environment.

What is PVM?

PVM is a software package that permits a heterogeneous collection of serial, parallel, and vector computers on a network to appear as one large computing resource.

Due to its widespread use, PVM has evolved from a system for implementing applications on networks of heterogeneous computers to an "industry standard" for parallel programming of homogeneous parallel computers. The experience with PVM stimulated and influenced strongly the development of the forthcoming Message Passing Interface (MPI) standard.

PVM/PARIX

PVM/PARIX is the implementation of the PVM public domain software package on PARIX systems from Parsytec, for example:

- MultiCluster–1, MultiCluster–2,
- SuperCluster,
- GCel,
- Xplorer.

PVM/PARIX is based on PVM 3.2.3. Currently, it supports PARIX 1.2 on Parsytec systems using the T800 family of Transputers and Sun4s as front–ends. PVM/PARIX consists of libraries of user–callable functions for applications written in C and FORTRAN, daemon programs which coordinate the inter–node activity, a console program which allows interaction with PVM applications, and tools which adapt PVM/PARIX to the user's specific system configuration.

What PVM/PARIX does

To use PVM/PARIX, the user has to break up his application into multiple "tasks" which cooperate to solve a problem in parallel. The tasks are loaded by PVM on different nodes of the PARIX system and are identified by task identifiers supplied by PVM. Each task independently performs a unit of computation and communicates by passing messages. Message passing is accomplished by calls to the PVM/PARIX libraries while the PVM/PARIX system takes care of routing the messages.

Highlights of PVM/PARIX

Heterogeneous parallel programming: PVM/PARIX supports fully heterogeneous parallel programming of networks of computers including Parsytec T800–based PARIX systems, Sun4s, and other systems running the public domain version of PVM. PVM/PARIX takes care of the necessary data conversions.

High bandwidth: The communication bandwidth of PVM/PARIX is close to the bandwidth of the underlying Virtual Links:

	PVM/PARIX		Virtual Links	
	Start–up time [μs]	Transfer rate [KBytes/s]	Start–up time [μs]	Transfer rate [KBytes/s]
node 0 to 1	1,960	952	28.5	1099
node 1 to 127	4,100	529	1,860	571

Timings for point–to–point communication have been obtained on a 512–node Parsytec GCel system. The higher set–up times are due to PVM's message handling and data conversion which are unavoidable on heterogeneous systems. The forthcoming homogeneous version of PVM/PARIX will reduce these start–up times significantly.

Optimized communication functions: Some of the PVM communication functions have been tuned for PARIX systems. The multicast routine `pvm_mcast`, for instance, sends a multicast message directly to affected processors of the PARIX system, thus, avoiding the overhead of routing the message through the daemon of the front–end machine.

Group communication: The PVM 3 group communication library is available with PVM/PARIX.

Additional tools: `pvmrun`, for instance, is the corresponding program to PARIX' `run` command. `pvmrun` starts PVM tasks on the PARIX system from the command line.

Using the documentation

The documentation of PVM/PARIX consists of the following documents:

- Introduction to PVM/PARIX (the document you are reading)
- PVM/PARIX Installation Guide
- PVM/PARIX: Design and Implementation Description
- PVM 3 Users's Guide
- PVM/PARIX Reference Manual

The *system administrator* who is in charge of the installation of PVM/PARIX should follow the instructions of the PVM/PARIX Installation Guide carefully.

Experienced PVM 3 users are adviced to study the Installation Guide to learn about the implementation specifics of PVM/PARIX. In addition, it is recommended to look at the manual pages which have been added or changed with respect to the public domain version of PVM 3. A list of manual pages with modifications indicated can be found at the beginning of the Reference Manual.

Novice PVM 3 users should start with reading the PVM 3 User's Guide which is taken from the public domain version of PVM 3. This will help to gain an understanding of the principal concepts of PVM 3 and the way how to write parallel programs with PVM 3.

Then, they should proceed to the Installation Guide to learn about implementation specifics of PVM/PARIX.

Users who are interested in the technical backgrounds of the PVM/PARIX implementation are refered to “PVM/PARIX: Design and Implementation Description”.

Appendix II PVM/Parix Design and Implementation Description

5 Introduction

This document provides a short description of the design and implementation issues for PVM/PARIX. Discussed are what criteria led to which decisions, and how it was implemented.

6 Design description

The heterogeneous version of PVM for PARIX was designed to be totally compatible to existing network–PVM implementations, yet to be efficient when working just on a PARIX system.

PVM wants point to point communication, without knowing about any topology. It does not provide its own routing. It's send operation is non–blocking, the receive can be done both blocking and non–blocking.

To be fast in communication, the choice was made to use virtual links. Virtual links provide blocking communication, and that is not always what PVM wants, as noted above. To provide non–blocking communication, threads have to be used to handle the communication, while the main code keeps on executing. Since the failure of one node to receive a message should not lead to a stop in communication from the node wanting to send to it, the choice was made to provide a thread for each virtual link.

PVM wants to be able to send/receive directly to any task. PVM provides no routing itself, and doing the routing on top of PARIX, which already has routing in it, is certainly not optimal. Using a totally connected network of nodes (via virtual links) was the one solution that solved this problem. One disadvantage of this approach is that the large amount of links needed takes up a lot of memory when using large numbers of nodes.

In the standard code of PVM version 3, some basic support for multiprocessor systems was already available. It was based on asynchronous communication, and it implemented the necessary buffering that provided compatibility with messages from other host types. These messages may arrive via, for example, a local network, and are broken in to pieces by the network–PVM code. This message disassembling and reassembling is quite an overhead, but unavoidable in the heterogeneous implementation.

The asynchronous communication wanted by the standard PVM 3 code was not readily available in the PARIX libraries (it needed, for example, to know on a per–message basis whether communication was done). This is why an extra communication layer was added. This layer also provides the necessary transparent communication from nodes to the front–end machine (the front–end is seen as another node by the layer).

The library code of PVM 3 uses some global internal buffer structures, and is not suitable for multi–thread usage. The heterogeneous version of PVM does not provide explicit placing of tasks on multiprocessor nodes, so the possibility of using multiple PVM tasks on one node did not seem very advantageous. This is why no support is provided for multiple PVM tasks on one node in the heterogeneous version.

The standard code of PVM 3 handles a multiprocessor machine by running a special version of the PVM daemon on the front–end. This daemon handles all requests for the multiprocessor. The rest of the PVM system only sees one machine through this daemon.: the multiprocessor. The daemon wants to be able to handle requests, and put tasks on each individual node on the multiprocessor machine. In PARIX, executing code from on a specific node is initiated from the node itself, by using the Execute() call. To mimic an execute that is initiated from the front–end (the daemon, to be exact), a small server has to run on each node, listening for requests from the daemon.

The next section will describe how the choices made were implemented.

7 Implementation description

The extra communication layer was implemented to provide the following:

- asynchronous communication
- communication to every other node
- transparent communication to the front–end machine
- use message handles which can be checked for the arrival of a message, whether it has been sent, who it was from, and what length it has.

This layer works on top of a fully connected set of nodes, i.e. each node has a virtual link to every other node. Each link is controlled by 2 threads: one to receive and one to send. Both threads use a linked list of requests and already received message for buffering. Semaphores are used to signal the arrival of messages and the availability of messages to send. Semaphores are also used to protect data structures against race conditions.

Communication to the front–end, is done by sending a message on an additional virtual link. This link connects the node to a control node. This control node has a connection to the front–end (using sockets). On a control node, 2 extra threads run: one to pass on messages from a socket to the nodes, and one to pass on messages from the nodes over a socket to the front–end. There is one control node in 32 nodes.

The PVM daemon listens to the set of control sockets coming from the nodes for messages, and passes on messages for nodes on them. A protocol also exists to pass on signals and execute commands from the front–end, using these sockets.

Some PVM calls have been optimized for speed. The multi–cast call normally contacts the daemon on the front–end to obtain a multi–cast address (a special identifier that is recognized by receiving tasks). Then it sends the message to the daemon, and the daemon redistributes it. This is a slow method of working. This is why the multi–cast in PVM for PARIX is implemented differently. It checks if the multi–cast only goes to tasks which are on the multiprocessor machine, and, if so, transmits the messages directly, thus bypassing the slow route via the host.

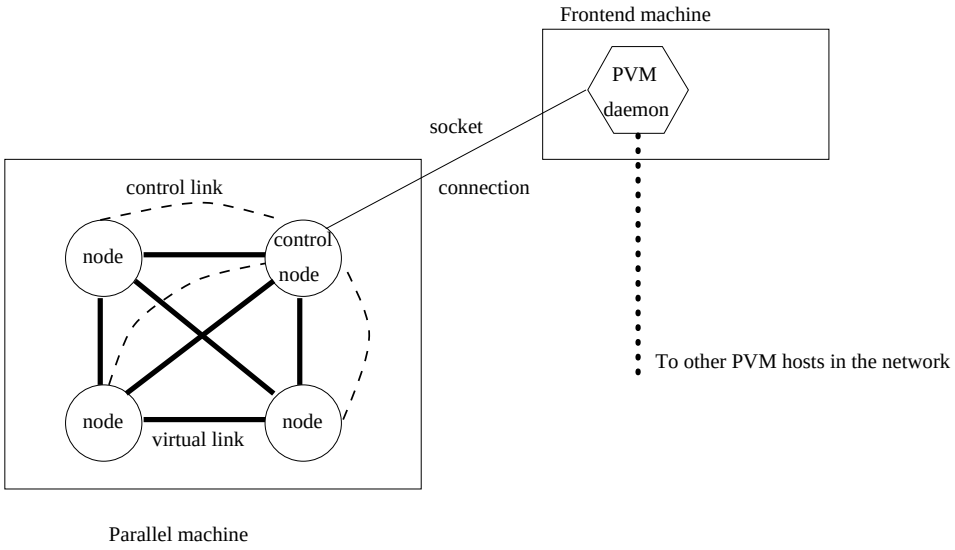
XDR is not available in the PARIX libraries. However, the source code for this from Sun is freely available, so it has been ported to PARIX, and is a part of the PVM library.

Group communication is implemented in PVM using a group server tasks (which is not optimal, and it will be done in another way in the homogeneous version, as will many other things). This group server was normally run on the front–end machine. This is obviously slow, and PVM/PARIX runs the group server on one of the nodes. This has the disadvantage that one node is always tied up by the group server.

When PVM is started, the daemon is started on the host. It does some internal initializing, and then allocates the nodes on the multiprocessor machine. Subsequently, the server binary $\$PVM\_ROOT/lib/PARIXT8/loader.px$ is loaded on to all nodes. This binary contains all code necessary to implement the network of virtual links and control nodes, and the extra communication layer. PVM binaries that are executed by the server, reference data structures set up by the communication code present in the server.

The server has one main loop, in which it listens for a command from the daemon on the front–end (the most important one is the execute command). When an execute command comes in, it starts a thread, in which the wanted binary is executed. The server waits for the thread to finish, and resumes the main loop.

Communication connections in PVM/PARIX.



Point-To-Point Communication, PVM/PARIX vs. Virtual Links