A Parallel Distributed Environment for Pakistan

Misbah U. Mirza, Asim-ur-Rehman, Zubair A. Shaikh, Ovais A. Khan
National University of Computer and Emerging Sciences
st-4,sector 17-D, Shah Latif Town on national Highway, Karachi, Pakistan
Email:misbah.mirza@nu.edu.pk

Abstract Parallel distributed computing systems provide mechanisms for exploiting parallelism inherent in many scientific and engineering applications. One such programming environment that has successfully demonstrated operation on a collection of heterogeneous computing elements incorporated by one or more networks is the Parallel Virtual Machine (PVM). It has been used on high end computing resources such as mainframe computers, multiprocessors, hyper cubes, and the like. In Pakistan, the most common computing resource is a low cost PC. The abundance of such machines provides an opportunity to develop and use a "poor man's supercomputer".

In addition, research on PVM has focused on Unix or similar platforms. None of the formal results, to evaluate certain benchmark applications, are available on Window based environments. The paper reports the results of the local PVM implementation and compares them with results from conventional implementations of PVM.

Keywords: Parallel Distributed Computing, Concurrent Programming, and Heterogeneous Computing.

1. Introduction

The introduction of Unix operating system provided an opportunity to programmers to fragment their applications in processes: a process acting as a separate program in its own rights. This has several advantages on serial machines, the most important being to encapsulate different tasks in a virtual machine environment. This meant they could be run and debugged separately without causing harm to the other parts of the application.

While Unix provided constructs to define processes and communication and synchronization among them, there was a justifiable need for providing a programming environment which could not only hide the complexity and existence of network communication (which is essential if different parts of an application are running on different connected computers) but also take advantage of the concurrency inherent in these applications by running those tasks in parallel. Hence a number of projects were started [1][2][3], which could result in such an environment. PVM (parallel virtual machine) was one such successful effort.

Unix, and its subsequent look-a-likes such as Linux, therefore provided a very useful abstraction of process to computer scientists which proved to be a building block for more elaborate parallel/concurrent programming environments, As a result a lot of research in parallel computing evolved out of it. In such a system processes can, in principle, be run in parallel on different machines connected together. However, computers could be of various makes, have different operating systems, and network types.

Hence, the PVM project aimed at providing a uniform transparent programming environment which could be supported by all these types of hardware and software. The advantage of such an approach was its cost effectiveness. Since many organizations already posses large quantities of low cost workstations PVM can provide a virtual parallel computing environment on top of them, hence enabling them to run large applications which could otherwise require large and expensive computers (and hardware).

Since its successful introduction into the academic community, PVM project has generated lot of new ideas in its application as well as extension. It has been ported to different for different languages like C++, and Java as well as script languages like Perl, and Tcl/tk. Some other projects have sprung off from the PVM project, like the Cumulvs environment [6] to support interactive visualizations for distributed applications running under PVM.

From its start, PVM was heavily influenced by Unix and Linux type of operating systems. Therefore, its first and foremost application was on systems having that kind of platforms. Since, the advent of the IBM PC, workstations have become more of a PC based machine, providing window based operating systems. Microsoft has emerged as the dominant leader in PC based workstation software platforms and MS windows (with its variants) being the operating system of choice for such machines
.

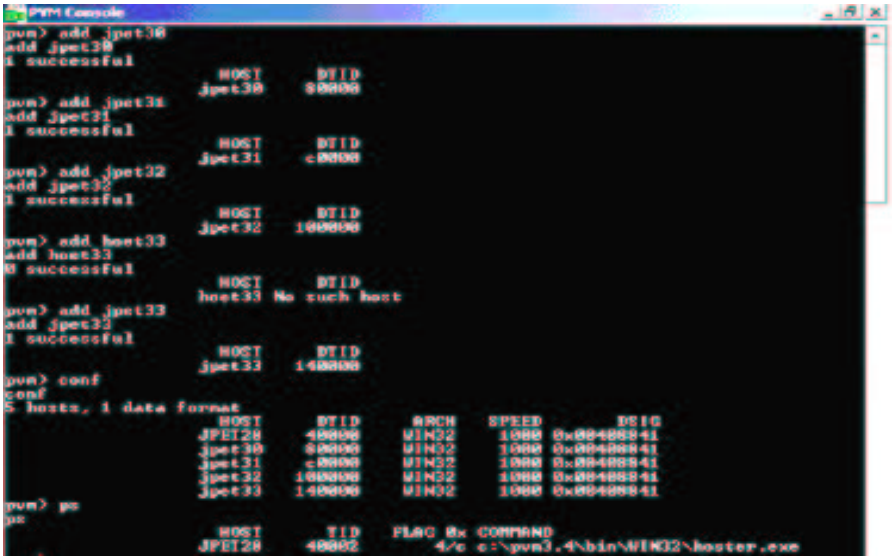

Figure 1. A typical PVM session

These workstations and operating system software are most economical and versatile in their application in the industry and academia (more so in the developing and third world countries like Pakistan, India, and the lot). This provides tremendous opportunity for tapping a resource which is so widely available in most local organizations and does not require any substantial expenditure. This has been the motive in extending PVM to support windows-based architectures [5].

However, the port of PVM to windows architecture has not been effortless or smooth. Most of PVM code has been written for Unix or Linux environment. Not all of it has been ported. In fact, only the user interface part of PVM has actually been ported for windows architectures. Therefore, the PVM support for windows is prone with errors. This research is concerned with porting PVM to windows architectures and networks available at local academic institutions in developing countries. The aim is to provide supercomputing resources at extremely low costs to the academic community which doesn't have the necessary financial resources to afford expensive supercomputers

This paper is structured into several sections. Section 2 discusses the setting up of the Windows-based PVM implementation. Section 3 describes running applications on such an implementation. Section 4 presents the performance evaluation of the implementation. Finally section 5 lays out the conclusions derived from the results of the experiments.

2. Setting up PVM on Windows-based PC clusters

Parallel distributed processing is beneficial for providing supercomputing for applications in the scientific and engineering domains. Providing this kind of prowess requires hundreds of millions of rupees. Such financial resources are not easily available in the developing world especially for educational institutions like technology colleges or universities. However, these institutions have extensive PC based laboratories for teaching and programming purposes. Today's low cost hardware and software for PCs have made resource sharing for communication among PCs over LAN/WANs and internet very cost effective even for these institutions.

Thus PVM provides an alternative of providing supercomputing affordable and readily available resources. These institutions have high end PCs and high speed LANs. These can be set up to support PVM and make the parallel programming instantly available for teaching and research staff.

Setting up PVM requires an examination of the prevailing operating system platforms. Majority of the PCs being used in academic institutions in Pakistan are Intel based running windows operating systems. Occasionally, there are some Unix/Linux servers also available. The need therefore was to find PVM libraries which could be setup in windows environments. Luckily, PVM project extensions have resulted in a PVM release PVM3.4 for windows[4][5].

Running PVM requires two software processes: master daemon and slave daemon. As windows environments do not

support Unix style 'forking', a separate 'hoster' process is required to do that. Hence the windows implementations of PVM need to provide three process.According to the virtual machine configurations of PVM, one master and one hoster process is required on the host machine. In addition, there is one slave daemon process for each additional computer in the virtual machine.

The PVM set up requires that remote execution mechanism be present on the participating computers. Unix has built in support for this in the form of rsh and rexec. However, windows platform used (Windows XP) did not provide this support. We decided to write our own remote shell daemon instead of relying on commercial packages so that we could experiment with it rather than relying on commercial packages.

As a result, an rsh daemon was developed and used successfully in our windows-based PVM implementation. This daemon provides the remote execution facility on a PC connected to the LAN but also caters for the security needs of the users owing the PC. This is essential to protect the PC from other people who can intentionally access the PC resources via the network. Hence, it provides a secure remote execution by restricting access to specific users and specific commands (such as allowing commands like rexec or rsh but not allowing del, dir, and cd commands).

To provide a convenient way for the users to interact with the PVM system, a PVM console process is used. From here, the user can launch parallel PVM applications. He/She can monitor the PVM performance and alter or reconfigure PVM environment by adding or removing computers from it. Figure 1 shows a typical PVM console session.

3. Running applications on the Windows-based PVM implementation

When applications run on PVM, they perform work by subdividing it and giving the work components to slave tasks. Slave tasks are created dynamically by calling PVM library functions which provide task creation capability to applications. Unlike Unix, windows based PVM implementation has to live with the fact that Unix like forking is not supported here. Instead, new tasks can be created and give the code to execute only from executable files. Therefore, the slave task code has to be separate file which has been compiled from different source files.

The common mode of application execution on PVM is to compile the application source code (for the master and the slave tasks). The compiled and linked files of the slave tasks are copied into the disk space of the slave computers (computers running the slave tasks). Then the application is run by executing it from the PVM console. This launches the master task which then creates slave tasks and activates them.

Once created tasks perform their work and return results to their parent task. This requires data communication among tasks. For this purpose, the PVM provides a message based communication protocol. PVM library provides mechanisms to broadcast messages as well to groups of tasks. This message protocol is completely hardware independent and the application or user sees it as a pure data transfer mechanism ignoring the hardware characteristics such as types of networks or operating system platforms. For complete introduction or reference to PVM message protocol see [2] or [3].

4. Performance of the Windows-based PVM implementation

To evaluate the effective performance and utility of the windows based PVM implementation, we looked at various benchmark applications that have been tried earlier during the PVM project.

We chose matrix multiplication as the trial application for two reasons. Firstly, it is most well understood application of parallelism. Secondly, there are parallel algorithms available for these tasks which are highly suitable for exploiting parallelism inherent in such applications. Once the effectiveness and usefulness of PVM under windows is proved we can turn our attention to other types of applications also.

A well known method of matrix multiplication, which makes itself very suitable for parallel execution, is to partition one

```
rows = Matrix Size / Number of Processors
for all Pl where 0 <= l < n do
   for i ← l to l + rows -1
      for j ← 0 to colsB
      C [i][j] ← 0
      for k ← 0 to colsA
         C [i][j] ← C [i][j] + (A [i][k] * B [k][j])
      endfor
      endfor
   endfor
endfor
```

Figure 2. A row-column oriented parallel matrix multiplication pseudo code

matrix into blocks of rows, assigning each block to a separate computing node. Then the second matrix is broadcast in full to each computing node. Now each computing node uses its block of the first matrix to multiply with the whole of the second matrix, calculating its block of the resultant matrix. This algorithm is shown in Figure 2.

No. of nodes	Matrix order			
	128	256	512	1024
1	0.771	9.153	73.155	590.579
2	0.871	5.217	38.515	302.234
4	0.931	3.305	22.252	163.375
8	0.871	2.424	13.529	91.221

Table 1. Results in seconds for the matrix-multiplication algorithm given in fig 2 under PVM for windows.

In such applications, matrix order determines the problem complexity as it effects the problem size. When we increase the matrix order, we increase the problem size, hence the problem complexity increases. We created PVM specific version of the algorithm in C++. After compiling it, we ran it several times on PVM, each time varying either the order of the matrix, the number of computing nodes, or both. Table 1 gives the results in seconds for the above algorithm when run in our windows-based PVM environment.

The above results were plotted (performance against problem size i.e. matrix order). Figure 3 below shows the resulting graphs. Each curve on the graph corresponds to the performance of the algorithm for the specified number of computing nodes when the problem size (matrix order) is varied.

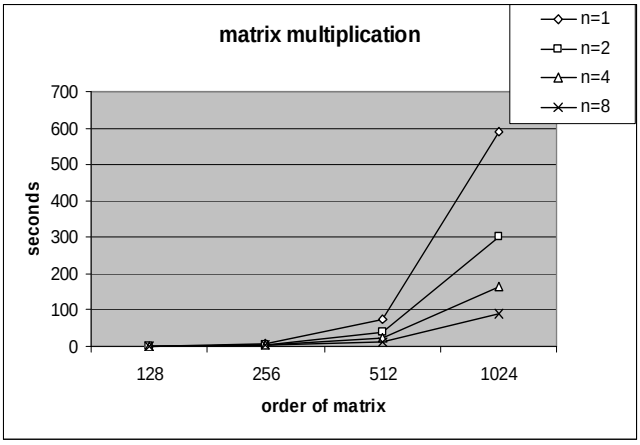

Figure 3. Graph showing the above performance figures when plotted agains problem size. Here 'n' is the number of computing nodes.

The graph clearly shows significant improvement in the performance of the algorithm for larger problem sizes when we increase the number of computing nodes involved in the running of the algorithm. Therefore, the implementation scales well when the matrices are large. For smaller matrices however, the locking and shared memory access overheads actually increase the execution times when more computing nodes are added. This result of the windows-based PVM implementation bears strong correlation with the general PVM performance results reported in [1].

Thus we have demonstrated when the problem size is increased on PVM under windows, adding more computing nodes to the virtual machine increases the computing power and hence better performance.

5. Conclusions

The main reason for implementing PVM on windows-based PCs has been the their wide spread availability. This paper presented such an implementation. The performance of the implementation was evaluated by running applications in parallel.

The results of the computations on this implementation demonstrated the usefulness of this approach by showing significant improvement over execution times for larger sized problems. More important, in situations like the ones we have in Pakistani academic institutions, is the ability of the PVM to utilize the resources that already exist and would be wasted otherwise. Other benefits could be the availability of a programming tool for new algorithms and applications.

PVM has provided a test bed for future ideas in parallel and distributed computing. As a result, it has been extended and formed the basis for further projects. The Harness project [7] is an example in this direction. It uses PVM to provide a virtual machine which is then used to support reconfigurable and scalable parallel distributed architectures.

On our part, we intend to pursue PVM application, and customize it so that we can explore some new ideas on such an environment, for example, support for developing and running mobile software agents on PVM.

References

1. V. S. Sunderam, *PVM: A Framework for Parallel Distributed Computing,* Concurrency: Practice and Experience, 2, 4, pp 315--339, December, 1990.
2. A. Geist, A. Beguelin, J. Dongarra, R. Manchek, W. Jiang, and V. Sunderam, *PVM: A Users' Guide and Tutorial for Networked Parallel Computing,* MIT Press, 1994.
3. A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam, *A Users' Guide to PVM Parallel Virtual Machine,* Oak Ridge National Laboratory, ORNL/TM-12187, September, 1994
4. G. Geist, J. Kohl, R. Manchel, and P. Papadopoulos, *New Features of PVM 3.4 and Beyond*, PVM Euro

Users' Group Meeting, pp 1-10, September, 1995, Lyon, France, Hermes Publishing, Paris.

5. Markus Fischer and Jack Dongarra, *Another Architecture: PVM on Windows 95/NT,* Concurrent Computing Conference, Atlanta, GA, March 10-11, 1994.
6. G. A. Geist, J. A. Kohl, P. M. Papadopoulos, *CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications*, International Journal of High Performance Computing Applications, Volume 11, Number 3, August 1997, pp. 224-236.
7. Mauro Migliardi and Vaidy Sunderam, *The Harness metacomputing framework*. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio (TX), USA, March 22-24 1999.