Volume 2, Issue 3, March 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Cluster Computing Using PVM

Rafiqul Zaman Khan
Department of Computer Science
Aligarh Muslim University
Aligarh (U.P), India
rzk32@yahoo.co.in

Md Firoj Ali
Department of Computer Science
Aligarh Muslim University
Aligarh (U.P), India

Abstract*— *When the execution time for a serial program is larger and there exists inherent parallelism in it then it is possible to design a parallel program that can be executed on more than one processor producing low execution time. In this paper, we represented the result of a serial matrix multiplication and the result of parallel matrix multiplication using Parallel Virtual Machine (PVM) using four PCs of dual core each. The result shows a better improvement on the performance when the program is executed in parallel. This implies that the bigger problem can be solved efficiently using PVM.

Key Words*— *Parallel Virtual Machine (PVM), Speed up and Efficiency

I. INTRODUCTION

Massively Parallel Processors (MPP) and Heterogeneous Computing (HC) are two important parallel and distributed computing paradigms. MPP is much faster than HC. But HC costs much less than the MPP. If performance is more important, MPP is the first option. Parallel Virtual Machine (PVM) was developed for the purpose of efficient heterogeneous computing.

PVM is an integrated software tools and libraries that are mainly designed towards networks of workstations. The central notion to the design of PVM is virtual machine concept. Virtual machine is defined as the collection of heterogeneous computers connected by a network which appears to a user as a single large computation system [1]. So using the combined speed and storage of many computers, the large computational problem can be solved with more cost effectively. The PVM system has been used for applications such as molecular dynamics simulations, superconductivity studies, distributed fractal computations, matrix algorithms, and in the classroom as the basis for teaching concurrent computing. The PVM system consists of two parts. The first part is a daemon which is known as pvmd3 and simply known as pvmd that exists in all the computers making up the virtual machine. The second part of the system is a library of PVM interface routines [3, 5].This library holds the functionally complete user callable routine for message passing, spawning process, co-coordinating tasks and modifying virtual machines.

PVM is more preferable than other heterogeneous computing tools for the following reasons:

- It is freely available standard software
- It can manage the heterogeneity more efficiently than other software.
- Virtual machine concept empowers the quality of this software.
- It has the ability of process control, group control, error handling and message passing.

For the heterogeneous computing a cluster of four computers was built in the Department high speed LAN. PVM was installed in every system.

This paper is organized as installation and configuration of PVM, running both example program and own program, some problems and solution and the conclusion.

II. INSTALLATION of PVM

A. *Step1*

Download the software package from http://www.netlib.org/pvg3

We downloaded pvm3.4.6.tgz in the home directory i.e., /home/mpiuser where mpiuser was my user name.

B. *Step2*

Going to the terminal and unpacked the software in the home directory. The unpacking process will automatically

create directory named pvm3 in the home directory. Pvm3 directory contains twenty one directory and thirteen files.
Command: $tar zxvf pvm3.4.6.tgz

C. Step3

Opened the .bhrc file through terminal using vi editor and set the following lines in the file [3] and closed the file.

The .bhrc is a hidden file of course and can be vied as:

```
$home
$ls –a
$vi .bhrc
```

Going to the insert mode add the following lines as:

```
PVM_ROOT=$HOME/pvm3
PVM_DPATH= PVM_ROOT/lib/pvmd
Export PVM_ROOT    PVM_DPATH
```

D. Step4

Going to pvm3 directory ($cd pvm3) type make ($make). This would make pvm(the PVm console), pvmd3(the pvm daemon), libpvm3.a(PVM C/C++ library), libfpvm3.a (PVM Fortran library) and libgpvm3.a (PVM group library). All these files would be placed in the $/pvm3/lib/LINUX and pvmgs (PVM group server) would be placed in $/pvm3/bin/LINUX.

III. CONFIGURATION

A. Step1

Open .rhosts file in the home directory ($ vi .rhosts) and added the name of each computer as:

```
f1
f2
f3
f4
```

Where f1, f2, f3and f4 are the name of each node(computer) in the cluster of four computers.

B. Step2

Set the following environment variables in .bashrc file as:

```
export  PVM_ARCH='$PVM_ROOT/lib/pvmgetarch'

export  PVM_ROOT=  $HOME/pvm3/xpvm

export   PATH =$PATH:$PVM_ROOT/lib

export  PATH==$PATH:$PVM_ROOT/lib/$PVM_ARCH
```

C. Step3

Going to the pvm3 directory and again executed make command.

D. Step4

Restart the computer and type pvm in terminal in the home directory. If a prompt pvm> is got means pvm is successfully installed.

IV. COMPILING AND RUNNING A PROGRAM

There are numbers of programs present in a subdirectory example in pvm directory under the home director. For compiling any program suppose hello.c and hello_other.c simply type on the terminal as follow:

```
$aimk hello  hello_other
```

This will make an object code corresponding to the source code in the directory /home/bin/LINUX. Before running any program pvm should be active mode that means the command pvm must be executed. There are two ways of running a program: i) going to the directory LINUX execute the command ./hello or ii) going to the example directory execute the command pvm> spawn-> hello.

A. Compiling and Running Own Program

Create your own directory (say myprog) in the home directory. Copy the Make.aimk file to myprog directory from the example directory.Let a program named matrixmul.c be executed. Copy this program to the myprog directory. Make an entry about matrixmul.c to the Make.aimk file like the other program entry as hello.c program there.

V. PERFORMANCE MEASUREMENT

Mainly two important parameters are used in measuring the performance of a parallel algorithm.

A. Speed up

Speed up of a parallel algorithm is the ratio of execution time when the algorithm is executed sequentially to the execution time when the same algorithm is executed by more than one processor in parallel. Speed up [4] can be mathematically represented as: $\mathbf{S_p=T_s/T_p}$, where $\mathbf{T_s}$ is sequential execution time, $\mathbf{T_p}$ is parallel execution time. In ideal situation, the speed up is equal to the number of processor in parallel but it is always less than the ideal one because the other important factors in a cluster like communication delay, memory access delay reduces the speed up.

B. Efficiency

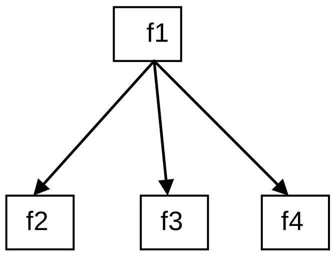
It is the measure of the contribution by the processors to an algorithm in parallel. Efficiency [4] can be measured as $\mathbf{E_p= S_p/p}$ $\mathbf{(0>E_p<1)}$ where $\mathbf{S_p}$ is the speed up and **p** is the number of processors in parallel. The Value of $\mathbf{E_p}$ is closure to 1 indicates an efficient algorithm.

VI. PARALLEL MATRIX MULTIPLICATION

Suppose, n is the number of rows of matrix A and p is the number of processors then Matrix A can be partitioned into n/p rows which will be assigned to the local-memory (LM) of each processor. Matrix B is made available to all the processors [3].

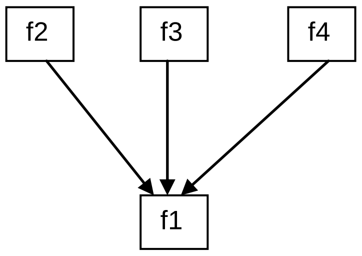
In distributed memory implementation, one processor (master processor) has both the matrix A and B in its LM. Master processor distributes the n/p rows of matrix A to each processor and broadcast matrix B to each processor and collect the result after the completion of computations [2, 3]. Fig 1 shows the data distribution and collection to and from the processors.

Data Distribution



n/p rows of matrix A and Matrix B are broadcasted by node f1 to f2,f3 and f4

Data Collection



Node f1 collected the results from f2, f3 and f4

Fig. 1 A Distributed memory implementation

VII. ANALYSIS of the RESULTS

From the Fig. 2 to Fig. 5 show that the performance of parallel algorithm depends on the number of processors in the cluster and on the size of the problem (matrix size in our case). The performance measurement parameters speed up and efficiency for parallel algorithm are shown in Fig 2 and Fig 5. There is a considerable reduce in execution time when the problem size is 1024 and the number of processor is eight. Again the fig 3 and fig 4 show the better speed up when the number of processor is eight the problem size is 1024. Fig 5 shows that the efficiency is highest when the number of processor is two and the problem size is 512 which mean the processor contributes its maximum effort that is up to 95%.

No of Processor(p)	Matrix Order		
	256X256	512X512	1024X1024
	Execution Time		
1	9.865	74.324	593.637
2	6.904	39.645	314.909
4	3.585	23.873	165.942
8	2.604	14.434	92.739

TABLE I
No. of Processor VS Execution Time

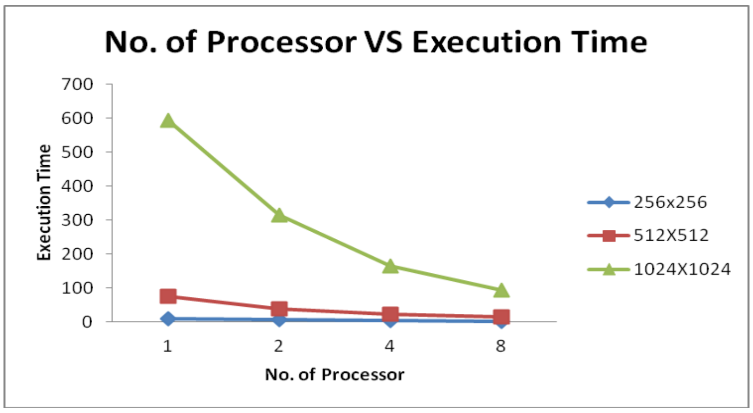


Fig.2 Number of Processor VS Execution Time

TABLE II
No. of Processor VS Speed UP

No. of Processor	Speed UP		
	265x256	512X512	1024X1024
2	1.42	2.75	3.79
4	1.87	3.11	5.15
8	1.89	3.58	6.40

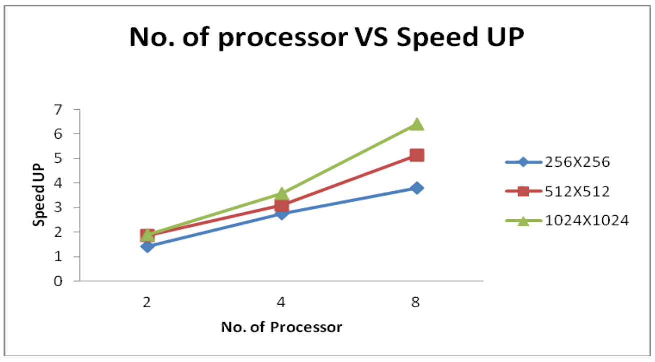


Fig.3 No. of Processor VS Speed UP

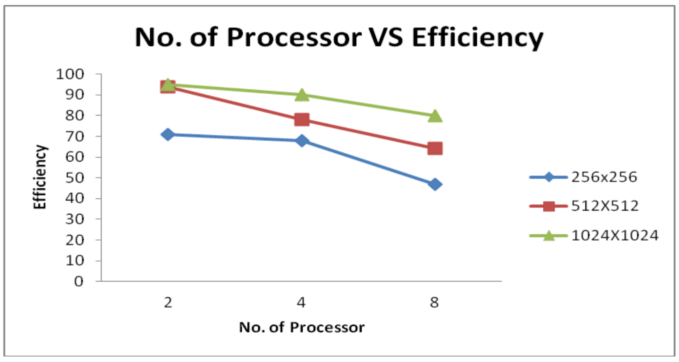


Fig. 5 No. of Processor VS Efficiency

TABLE III
Matrix order VS No. of Processor

Matrix Order	No. of Processor		
	2	4	8
256X256	1.42	2.75	3.79
512X512	1.87	3.11	5.15
1024X1024	1.89	3.58	6.40

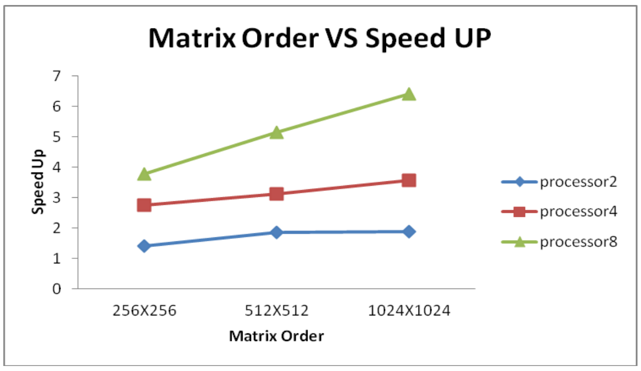


Fig. 4 Matrix order VS No. of Processor

No. of Processor	Efficiency (%)		
	256X256	512X512	1024X1024
2	71	94	95
4	68	78	90
8	47	64	80

TABLE I V
No. of Processor VS Efficiency

VII. CONCLUSION

This paper has presented parallel implementations of matrix multiplication using distributed-memory approaches. Matrix multiplication is an efficient algorithm for the measure of the parallel processing parameters as the execution time speed-up/efficiency as shown by the figures. The results of the computation show the considerable improvement over the execution time when the problem size is 1024 and the number of processor is eight. So we can apply PVM for solving even larger problem faster. The result shows a better improvement on the performance when the program is executed in parallel.

REFERENCES

1. Geist G. A., Kohl J. A., and Papadopoulos P. M., "PVM and MPI: A Comparison of Features”, May 30, 1996.
2. Gupta A. and Kumar V.”Scalability of Parallel Algorithms for Matrix Multiplication*”. TR 91-54, November, 1991.
3. Hussain J. S. And Ahmed G. “A Comparative Study and Analysis of PVM and MPI for Parallel and Distributed Systems”. IEEE, 2005.
4. Lee M.-C. “A Divide-and-Conquer Strategy and PVM Computation Environment for the Matrix Multiplication”. Springer-Verlag Berlin Heidelberg 2009.
5. Sunderam V. S. “PVM: a parallel framework for parallel distributed computing”. 1990.