Evaluation of Parallel Application’s Performance Dependency on RAM using Parallel Virtual Machine

Sampath S¹, Nanjesh B R¹
¹Department of Information Science and Engineering
Adichunchanagiri Institute of Technology,
Chikmagalur, Karnataka, INDIA

Pramod H B²
²Department of Computer Science and Engineering
Rajeev Institute of Technology, Hassan,
Karnataka, INDIA

Abstract

Parallel computing is the simultaneous use of multiple compute resources to solve a computational problem. Parallel computing operates on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). The reasons for using parallel processing are to save time (wall clock time), to solve larger problems and to provide concurrency while taking advantage of non-local resources and overcoming memory constraints. We aim to present a framework using PVM that demonstrates the performance dependency of parallel applications on RAM of the nodes (desktop PCs) used in parallel computing. This can be realized by implementing matrix multiplication problem on the framework. The framework consists of a client, a master, capable of handling requests from the client, and a slave, capable of accepting problems from the master and sending the solution back. The master and the slave communicate with each other using pvm3.4.6. The master will monitor the progress and be able to compute and report the time taken to solve the problem, taking into account the time spent in assigning the problem into slave and sending the results along with the communication delays. We aim to compare and evaluate these statistics obtained for different sizes of RAM under parallel execution in a single node involving only two cores, where one acts as master and other as slave. We also show the dependency of serial execution on RAM for the same problem by executing its serial version under different sizes of RAM.

Index Terms— Parallel Execution, Cluster Computing, Symmetric Multi-Processor (SMP), PVM (Parallel Virtual Machine), RAM (Random Access Memory) size.

1. Introduction

Parallel processing refers to the concept of speeding up the execution of a program by dividing the program into multiple fragments that can execute simultaneously, each on its own processor. This paper deals how to handle Matrix Multiplication problem that can be split into sub-problems and each sub-problem can be solved simultaneously. With computers being networked today, it has become possible to share resources like files, printers, scanners, fax machines, email servers, etc. One such resource that can be shared but is generally not, is the CPU. Today’s processors are highly advanced and very fast, capable of thousands of operations per second. If this computing power is used collaboratively to solve bigger problems, the time taken to solve the problem can reduce drastically. This paper is an attempt to determine exactly to what extent the approach of parallel computation can be used and to numerically evaluate the mechanism. It can also determine the change in problem solving time when more systems are added or systems are removed from the network.

1.1 Existing frameworks

1.1.1. PVM 1.0: PVM that have been released from the first one in February 1991. PVM 1.0 cleaned up the specification and implementation to improve robustness and portability [2].

1.1.2. PVM 2.X Versions: PVM 2.1 provided with process-process messages switched to XDR to improve portability of source in heterogeneous environments and simple console interpreter added to master pvmd. Later versions belonging to PVM 2.x provided with more and more useful functionalities such as pvmd-pvmd message format switched to XDR, get and put functions vectorized to improve performance, broadcast function deprecated, improved password-less startup via rsh/rcmd etc [2].
1.1.3. PVM 3.X Versions: These allow scalability to hundreds of hosts, allow portability to multiprocessors / operating systems other than Unix, allows dynamic reconfiguration of the virtual machine, allows fault tolerance, includes dynamic process groups, provide the option to send data using a single call [3].

1.1.4. PVM 3.4.6: Includes both Windows and UNIX versions and improved use on Beowulf clusters. Also includes the latest patches for working with the latest versions of Linux (like fedora 14), Sun, and SGI systems New features in PVM 3.4.x include communication contexts, message handlers, persistent messages. In our project, we are using PVM 3.4.6 for providing parallel environment using PVM [4].

1.2. Proposed System

This paper deals with the implementation of parallel application, matrix multiplication [5] on PVM framework using PVM 3.4.6 which is best suitable for LINUX based systems. The proposed system demonstrates the performance dependency of parallel and serial execution on RAM and also it demonstrates the following:

- How a client can submit the entire problem to a master and collects the solution back from it without bothering about how it has been solved.
- How the master detects the available slaves on the network, and how it detects the system load on that machine to determine whether it is worth sending a task to that particular client.
- How a problem can be submitted to the slaves.
- How the solutions of the given problem can be retrieved from the slave.
- How the slaves solve the given problem.

2. Related works

Traditionally, multiple processors were provided within a specially designed "parallel computer"; along these lines, Linux now supports SMP Pentium systems in which multiple processors share a single memory and bus interface within a single computer. It is also possible for a group of computers (for example, a group of PCs each running Linux) to be interconnected by a network to form a parallel-processing cluster [1].

V.S Sunderam, G.A Geist, J Dongarra, R Manchek (1994) [6] describe the architecture of PVM system, and discuss its computing model, the programming interface it supports, auxiliary facilities for process groups and MPP support, and some of the internal implementation techniques employed. Amit Chhabra, Gurvinder Singh (2010) [7] proposed Cluster based parallel computing framework which is based on the Master-Slave computing paradigm and it emulates the parallel computing environment. Muhammad Ali Ismail, Dr. S. H. Mirza, Dr. Talat Altaf (2011) [8] performed the Concurrent Matrix Multiplication on Multi-Core Processors. IKamalrulnlzam Abu Bakar, IZaitul Mlir lzawati Zal nuddln (2006) [9] made the performance comparison of PVM and RPC. The comparison is done by evaluating their performances through two experiments namely one is a broadcast operation and the other are two benchmark applications, which employ prime number calculation and matrix multiplication. We aim to present a framework using PVM that demonstrates the performance gain and losses achieved through parallel/distributed processing. And also demonstrates the performance dependency of parallel applications on RAM.

3. System requirements

3.1. Hardware Requirements

- Processor: Pentium 4 (3 G Hz)
- Two RAM: 256MB and 1GB
- Hard Disk Free Space: 5 GB
- Network: TCP/IP LAN using switches or hubs (if more than one node)

3.2. Software Requirements

- Operating System: Linux
- Version: Fedora Core 14
- Compiler: GCC
- Network protocol: Secure Shell (if more than one node)
- Communication protocol: PVM

4. System design

The design was made modular i.e. the software is logically partitioned into components that perform specific functions and sub-functions. Master is designed such that it has functionality to manage connection and communication with the slave, It scans and identifies all the cores or slaves available on the node here it is only one slave to be identified. It then assigns the processor ranks to identify the cores. The master assigns the problem to slave. It also has to accept the results sent back by the slave after they finish the computation of the sub-tasks assigned to them. Then the received result has to be assembled in the right order to obtain the solution for the main problem. Slave is designed to have the functionality to read the problem (in case of single slave)/sub-problem sent by the master, evaluate the problem (in case of
single slave)/sub-problem and send the result back to the master.

**4.1 Architecture for presenting Parallel Computing Framework.**

The main problem is taken by the master core and assigns the task into slave core. Slave core send back the solutions of the assigned task or problem.

**Figure 1. Architecture for presenting Parallel Computing Framework**

**4.2 Configuration**

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

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. Command: $tar zxvf pvm3.4.6.tgz

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 done as:

```bash
$home
$ls -a
$vi .bhrc
```

Going to the insert mode add the following lines as:

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

Going to pvm3 directory ($cd pvm3) type make ($make). This would make pvm (the PVm console ), pvmd3 (the pvm daemon), libpvm3.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.

Set the following environment variables in .bashrc file as:

```bash
export PVM_ROOT=`$HOME/pvm3`xPVM
export PVM_ROOT= HOME/pvm3/xpvm
export PATH=$PATH:$PVM_ROOT/lib
export PATH===$PATH:$PVM_ROOT/lib$/PVM_ARCH
```

Going to the pvm3 directory and again executed make command. If a prompt pvm> is got means pvm is successfully installed.

**5. Implementation**

Implementation is the most crucial stage in achieving a successful parallel system. The problem to be solved has to be parallelized so that computational is reduced. The framework consists of a client, a master core, capable of handling requests from the client, and slave, capable of accepting problems from the master and sending the solution back. The master and the slave communicate with each other using PVM 3.4.6. The problem has to be divided such that the communication between the server and the client is minimum. The total computational time to solve the problem completely is effected by the communication time between the nodes.

**5.1 Parallel Matrix Multiplication Design**

In the algorithm which we have implemented is for solving matrix multiplication problem on several nodes it may be for only one or more slaves. It divides the matrix into set of rows and sends it to the slaves rather than sending one row at a time [6]. The slaves compute the entire set of rows that they have received and send it back to the server in one send operation. Hence, we need to implement parallel systems consisting of set of independent desktop PCs interconnected by fast LAN cooperatively working together as a single integrated computing resource so as to provide higher availability, reliability and scalability. But to show the performance dependency on RAM we are considering only single node with two cores, one acts as master and other as slave. So there will be no division of problem, instead entire problem is submitted to the single available slave. Consider two matrix, matrix A and B.
The flow of multiplication of matrix A and B takes place as shown in Fig. 2.

![Flow Diagram for Solving Matrix Multiplication Problem on Several](image)

Figure 2. Flow diagram for solving matrix multiplication problem on several.

Each slaves is sent one set of rows of the first matrix A and the entire second matrix B and the slave send back a set of rows of the resultant matrix. The master assembles the rows of the resultant matrix in the correct order. Fig. 3 shows how the matrix multiplication is designed.

![Parallel Matrix Multiplication Design](image)

Figure 3. Parallel matrix multiplication design

6. Testing

6.1. Testing of Small Order Matrices

We have tested the output of our system by verifying the output using a online matrix calculator. We have repeated this test for different smaller sizes of matrix. The tool used by us is Blue Bit online matrix multiplication calculator.

6.2. Testing of Higher Order Matrices

Blue Bit online matrix multiplication calculator is limited to matrix of order 32 as the testing tool does not support the matrix multiplication of higher order. To verify our result for higher order matrix we have assigned 1 to all the elements of the matrix. The elements of the output matrix will be equal to the order of the matrix. Example is shown in Fig. 4.

7. Results and Analysis

We have analyzed the performance of parallel method against traditional serial method. The results are tabulated and compared. We calculated the time for solving the matrix multiplication problem using both serial and parallel algorithm. From the Table I we can conclude that Performance of serial execution almost remains same even after the increase in RAM size. There are negligible computation time variations for increase in RAM size. This is because the Serial execution is performed by the cores itself with negligible RAM usage and also due to the no communication involved between cores. Hence it is independent of RAM. From the Table II we can conclude that Performance of parallel execution drastically increases when there is increase in RAM size. It shows drastic decrease in computation time with the increase in RAM. Because parallel execution oftenly uses RAM for the communication between cores and also it involves lot of send and receive operations and temporarily storing the result of problem assigned to cores. Higher the size of matrices the time difference in Table II is very high, because higher the matrix size, more will be sends and receives resulting in the need of higher utilization of RAM. So for smaller RAM the computation time will be more and larger the RAM size, computation time will be less in parallel execution finally resulting in better performance.

Generally we can say parallel execution is faster than the serial execution but the results of serial execution with 1000MB RAM and parallel execution with 1000MB RAM shown in the Table 1 and Table 2.
### TABLE I

Comparison of results of serial execution on different RAM size

<table>
<thead>
<tr>
<th>Size of the matrix</th>
<th>100*100</th>
<th>500*500</th>
<th>1000*1000</th>
<th>1500*1500</th>
<th>2000*2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial execution with 256 MB RAM</td>
<td>0.0159 seconds</td>
<td>1.2744 seconds</td>
<td>9.8071 seconds</td>
<td>32.9938 seconds</td>
<td>78.1319 seconds</td>
</tr>
<tr>
<td>Serial execution with 1000 MB RAM</td>
<td>0.0157 seconds</td>
<td>1.2713 seconds</td>
<td>9.8112 seconds</td>
<td>32.9798 seconds</td>
<td>78.1452 seconds</td>
</tr>
<tr>
<td>Time difference</td>
<td>-0.0002 seconds</td>
<td>-0.0031 seconds</td>
<td>+0.0041 seconds</td>
<td>-0.0014 seconds</td>
<td>+0.0133 seconds</td>
</tr>
</tbody>
</table>

### TABLE II

Performance dependency of parallel application on RAM

<table>
<thead>
<tr>
<th>Size of the matrix</th>
<th>100*100</th>
<th>500*500</th>
<th>1000*1000</th>
<th>1500*1500</th>
<th>2000*2000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel execution with 256 MB RAM</td>
<td>0.3752 seconds</td>
<td>8.3144 seconds</td>
<td>46.5224 seconds</td>
<td>769.1427 seconds</td>
<td>1085.2721 seconds</td>
</tr>
<tr>
<td>Parallel execution with 1000 MB RAM</td>
<td>0.1951 seconds</td>
<td>6.8115 seconds</td>
<td>26.9792 seconds</td>
<td>76.2485 seconds</td>
<td>170.6431 seconds</td>
</tr>
<tr>
<td>Time difference</td>
<td>-0.1781 seconds</td>
<td>-0.4049 seconds</td>
<td>-19.5432 seconds</td>
<td>-692.8942 seconds</td>
<td>-914.629 seconds</td>
</tr>
</tbody>
</table>
Figure 4. Output of 1000 * 1000 matrix in parallel execution

respectively shows that serial execution is faster than parallel execution in a single node having two cores, for different sizes of matrices. This is due to the communication overhead involved in the parallel execution but this can be overcome by increasing the number of nodes but at present it is out of scope of our work and can be done as future work. Overheads that are considered are the connection time required to connect to slave, time taken to send the problem along with inputs to slave time taken to retrieve the solutions from the client, time taken to assimilate the results obtained.

8. Conclusion

We developed a framework that demonstrates the performance gain and losses achieved through parallel/distributed processing. Matrix multiplication problem is solved serially and also in parallel. We presented a framework using PVM that demonstrated the evaluation of the performance dependency of parallel applications on RAM. We also showed the dependency of serial execution on RAM for the same problem by executing its serial version under different sizes of RAM.

9. Future works

We compared the results with runs on single node only. It can be extended to the more number of nodes to evaluate the performance dependency on RAM with the increase in number of nodes using same parallel matrix multiplication algorithm. Even though the method that has been used here can be deployed to solve larger order problems, it is cumbersome to give the data input for matrices of larger order. Hence this work can be extended to give input from files for larger order matrices. It can also be extended to solve other similar problems related to matrices, like finding the determinant and other backtracking problems. The analysis is also useful for making a proper recommendation to select the best algorithm related to a particular parallel application. If the nodes are extended, node failure can be a problem that has to be tackled.

References


