Journal of Engineering and Applied Sciences 12 (Special Issue 2): 6313-6317, 2017
ISSN: 1816-949X

Performance Evaluation of Parallel Applications using SVM and PVM Through Matrix Multiplications

[1]Mehzabeen Kaur and [2]Surender Jangra
[1]Department of Computer Science and Engineering, BBSBEC, Fatehgarh Sahib, Punjab, India
[2]Department of Computer Science and Applications, GTB College, Bhawanigarh, Sangrur, Punjab, India

Abstract: Parallel computing environment is playing a major role in high performance computing which provides performance similar to that of mini supercomputers in a cost effective way to solve such a computation intensive applications. With the continuous developments in node architectures and interconnection technologies, MPI and PVM are creating the space for significant role in building up parallel programming environment. In this study an attempt has been made to evaluate the performance of parallel applications using SVM and PVM by resorting to large order metrics multiplications as a benchmark.

Key words: MPI, PVM, serial, parallel computing, matrix multiplication, benchmark

INTRODUCTION

The demand for high computational power is rising dramatically in the present technical scenario for solving engineering and scientific complexities. Due to the lack of economical and other reasons supercomputer is not approachable for a common human being. The Information Technology (IT) revolution has really been the miraculous approach of the first and second decades of the twenty first century and a lot more untouched areas are being explored to further make the technological jump. There are most of the computation power gets wasted without taking into consideration of different desktop PC, laptop and mobile phones. It may also lead to some other areas where we can think afresh and work accordingly to delve deep into technological advancements. Easily availability of LAN based networked PC and high network bandwidth make it possible to design such a platform through which we can enhance the computation power with a lower cost which are affordable to everyone.

Parallel computing uses multiple available resources to solve large problem by sharing their computational power which operates in such a way where large problems take the shape of smaller ones where the concurrent solving process helps in saving the time by resorting to more and more computing devices.

MPI and PVM play an important role for successfully implement the concept of parallel computing. Portability and heterogeneous nature of MPI and PVM, makes these two devices more system independent. There are many situations when one method of application is more effective than the other one. No device is perfect in its own way as PVM supports fault notification and dynamic in nature which provides virtualization but MPI is much advanced message passing communication, static in nature and provide high level abstraction. Use of MPI and PVM largely depends upon on the nature of the application and its suitability (Sampath *et al*., 2013).

The beginners in the field of parallel computing will know only parallel processing is better and faster than serial processing but they will not be thorough on what circumstances parallel performance overcomes the serial. This motivated us to begin our work with differentiate between of serial and parallel using both MPI and PVM, at different conditions. There is a need of knowing the dependency of serial and parallel processing on the RAM of anode to illustrate communication overhead incurred in parallel processing.

Literature review: Anderson *et al*. (1995) is more popular for parallel environment in respect to using the computation power through network. Boklund *et al*. (2005) is largely used for commercial applications likes cluster farms, Multi or distributed parallel processing environments. Chhabra and Singh (2010), suggests a cluster based parallel computing structure using PVM and MPI for different problems using socket programming. Ismail *et al*. (2011) found open MP compared with the matrix multiplication done concurrently on multi-cores. Many researchers are carried out on analysis of open MP

Corresponding Author: Mehzabeen Kaur, Department of Computer Science and Engineering, BBSBEC, Fatehgarh Sahib, Punjab, India

performance with respect to other tools. PVM based architecture and performance issues relating to different communication overheads are analysed by Sunderam *et al.* (1994). The proposed research includes case studies in environmental science, materials science and modelling of climate variations to show the feasibility of PVM for scientific applications. Similar research (Sharma *et al.*, 2011) on MPI performance analysis are carried out by Shan and Singh (2001). But, these works does not focused on the methodologies of assigning work load to the slaves. Cirtek and Racek (2007) suggests the framework consists of client and the servers which communicate with each other using MPICH-2, a portable implementation of Message Passing Interface (MPI) for coordination among the processes and to reduce the computation time for solving larger matrix multiplication problem. Khan and Ali (2011) performed cluster computing using PVM checked for matrices of size up to 1024×1024 but PVM for solving even larger problem is not shown.

Their results presents improvement through parallel but not compared with serial execution for same size of matrices. Mostaccio *et al.* (2006) analysed different alternatives for such simulation and their application to obtain performance and model scalability using different implementation to an individual oriented model. Meenal and Prashant (2014) show multi-core CPU's have given different impulse to shared memory model programming. Along with this, heterogeneous programming is explored, to exploit combined effects of CPUs and Graphics Processing Units (GPUs) graphics processing. Kumar and Jangra (2015, 2016), developed an automated fault tolerance framework in Alchemi Computation power based upon memory and CPU power. Kumar *et al.* (2008), Sejwal *et al.* (2012), Sampath *et al.* (2013) develop an fault tolerance architecture for mobile distributed systems.

MATERIALS AND METHODS

Experimental setup

System requirement and specification: System requirement specification involves the specific requirements, software and hardware requirements followed by the interfaces needed for the communication between the master and slaves. Hardware and software requirement on each node as under:

Hardware requirements:

- Dual core-Pentium 4 (3 GHz)
- 1 GB RAM
- 5 GB h disk free space
- TCP/IP LAN network using switches

Software requirements:

- Linux operating system Fedora Core 14 Version
- GCC compiler
- Secure shell network protocol
- MPICH2 and PVM 3.4 communication protocol

Simple C code for testing of matrices: Parallel applications can be written using Fortran, C and C++ languages with PVM or MPI routines embedded in code as functions or subroutines. The program is compiled for each machine available in parallel computing cluster and the resulting object files are made accessible to all the machines. In our research we use C programming with PVM or MPI routines embedded in code as functions or subroutines.

The simple C code used here is included in the Master program and to verify the resultant matrix. The size of input matrices is assigned same (MATSIZE) for both input matrices. The elements of the output matrix must be equal to the order of the matrix. This can be easily applicable to any size of matrices. When the master process receives all the elements of resultant matrix C, it verifies each element of resultant matrix whether it is equal to the order or size of input matrix (Algorithim 1).

Algorithim 1; Procedure to check for errors in resultant matrix:

```
count = 0
for i = 0 to NRA
  for j = 0 to NCB
  if c[i][j] == MATSIZE
       count = count+1; print c[i][j]
if count == MATSIZE×MATSIZE
              print count value as "number of elements checked for correct
         and are correct"
else
      print error
```

The overall time taken to solve the problem in both the cases MPI and PVM is measured which includes the time for breaking the problem, assigning the subtasks, receiving the solutions of sub tasks, consolidating the solutions and communication overheads state chart diagram of parallel computation shows various events and states of master and slaves that occurs when the user specifies the number of processes, input matrices size and start the master process.

Client server matrix multiplication process: Slaves moves from idle state to waiting state when they are started and wait for the input matrices. When a slave receives the rows set of matrix A and B, it enters the computing state and computes the part of the resultant

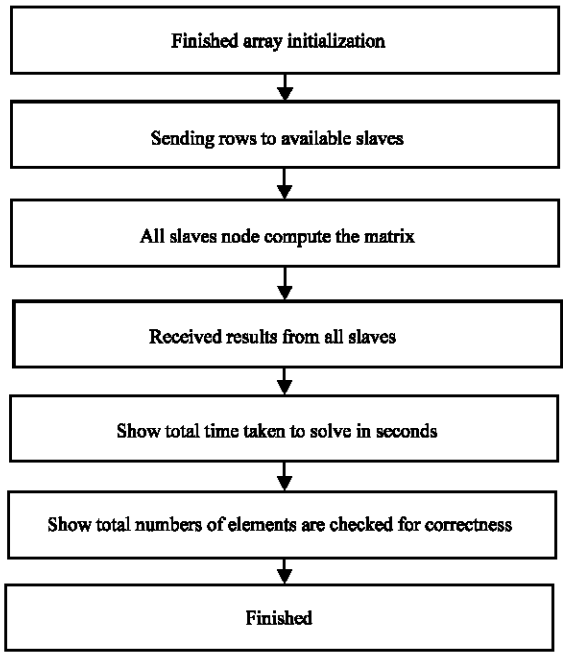

Fig. 1: Client server matrix multiplication process

matrix C. Then it sends back the result to the master and enters to the idle state when the send is finished (Fig. 1).

RESULTS AND DISCUSSION

Proposed parallel programming platform is checked for MPI as well parallel programming. We also check whether the parallel matrix multiplication is working properly over the single node as well multiple nodes and also check division of first matrix among the slaves. First, the test is made over single node then over multiple nodes for different size of matrices. Next part of testing is to check the resultant matrix for its correctness for large order matrix (Table 1).

Case 1; One node, two nodes and three node analysis with below 1000×1000 matrix: In case 1 the communication overhead is more than computation (Sampath *et al*., 2013) when the size of the matrix is small level. As master node has to be coordinate with all the executing node who are located over the different cores of nodes that are included for parallel computation. It means that two nodes are sufficient for computation and also it shows the capacity of single node is not sufficient to perform the computation with less time. If we increase the number of nodes more than the sufficient number of nodes it results in communication overhead.

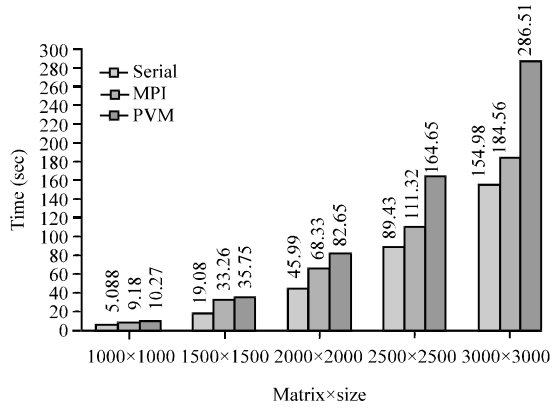

Fig. 2: Serial vs parallel computing time using one node with MPI and PVM

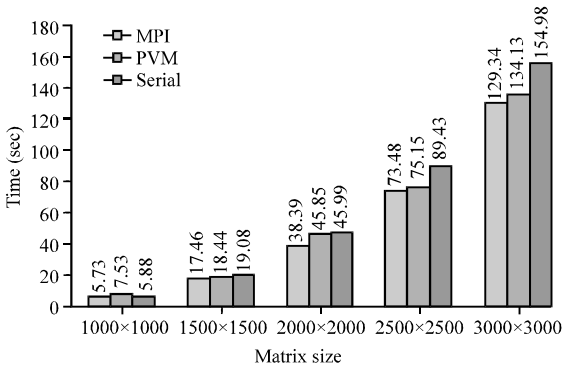

Fig. 3: Serial vs parallel computing time using two node with MPI and PVM

Table 1: Performance of serial and parallel execution for high order matrices (At 1GB RAM, all time in seconds)

Variables	1000×1000	1500×1500	2000×2000	2500×2500	3000×3000
Serial and parallel computing using one node					
Serial	5.88	19.08	45.99	89.43	154.98
MPI	9.18	33.26	68.33	111.32	184.56
PVM	10.27	35.75	82.65	164.65	286.51
Parallel computing using two nodes					
MPI	5.73	17.46	38.39	73.48	129.34
PVM	7.53	18.44	45.85	75.15	134.13
Parallel computing using three nodes					
MPI	5.28	14.91	31.25	56.82	97.62
PVM	7.66	19.13	36.28	66.96	108.26

Case 2; one node, two nodes and three node analysis with more than 1000×1000 matrix: As the size of the matrix increases (Table 1) multiple executor can done their job in very short time comparative to single one. Since, the computation will be more for larger matrices, it needs more number of executor depending upon the size of the matrix to give better performance. Hence, for larger matrices, we get optimal computation time over the three nodes as our work is limited to three nodes. Figure 2 and 3 the MPI and PVM, respectively.

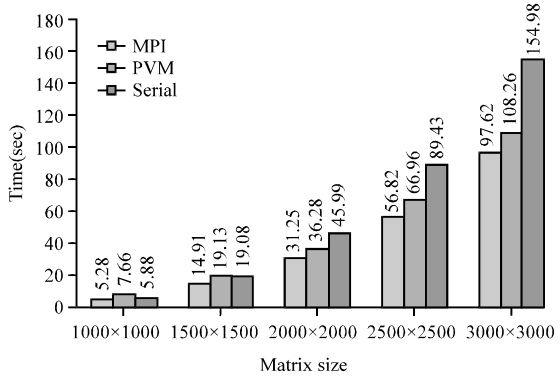

Fig. 4: Serial vs parallel computing time using three node with MPI and PVM

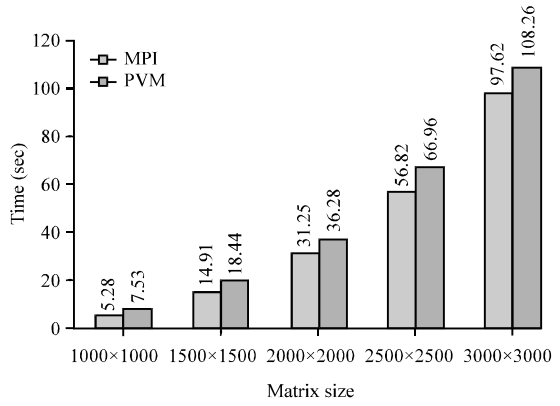

Fig. 5: Optimal time for different high order matrix multiplication

Case 3; Comparison of serial versus parallel execution with MPI and PVM: Table 1 shows the comparison of serial and parallel execution for higher order matrices. Sejwal *et al.* (2012), since, the matrix size is small, the computation is less; hence, there is no much time difference between the serial and parallel and serial shows better performance than parallel for smaller order matrices. This is because the communications between the processes decrease the parallel execution performance where as in serial there is no communication as the single core performs the execution sequentially (Fig. 4).

Figure 2-5 show the graphical representation of performance of serial versus parallel execution with MPI and PVM respectively for high order matrices. In Table 1 since the matrix size is large, there will be more computation resulting that serial is very much slower than the parallel execution especially for higher order matrices over three nodes. We can also observe that serial execution is faster than parallel execution over the single node especially for larger matrices.

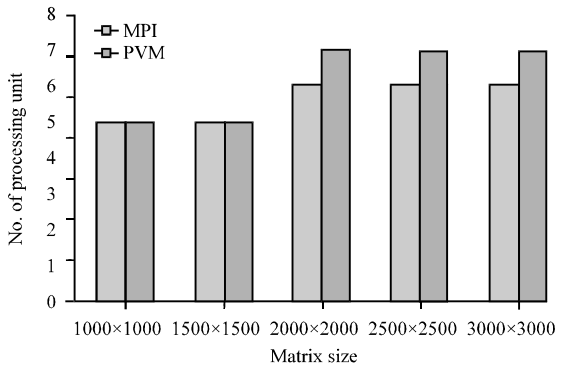

Fig. 6: Optimal number of executor in high order matrix multiplications

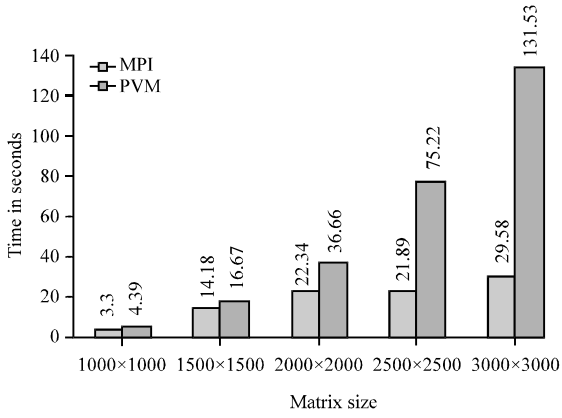

Fig. 7: Communication overhead incurred in MPI and PVM

Case 4; Optimal time and no. of computing resources required for MPI and PVM: Figure 5 shows the optimal time in seconds for difference high order matrix multiplication through parallel processing and further Fig. 6 shwos the optimal number of computing resources required. MPI shows their better performance in both the cases means minimum time and number of executing the matric multiplications.

Case 5; Communication overhead in MPI and PVM: Figure 7 shows the communication overhead incurred in both MPI and PVM. When we compares the communication overhead MPI have minimum overhead than PVM.

CONCLUSION

It is an attempt to implement the parallel matrix multiplication solution using MPI and PVM individually. The comparison of the MPI and PVM based on the computation time taken by them to solve matrix

multiplication and also compare with serial execution. Work also demonstrates the communication overheads, optimal time and number of computing resources required for MPI and PVM, optimal number of processing unit required for matrix multiplication through test cases. The work also involves finding the break-even point regarding optimal number of slaves required for the computation when number of nodes is fixed by the user under both MPI and PVM. It is concludes that MPI is faster than PVM for larger order matrices also and gives faster performance than PVM with the runs on increase in number of nodes.

Further, optimization approach of this research can be applied for other parallel applications also such as parallel merge sorting that can be used as alternative for parallel cursing in IT sector. It can be used in sorting of voters list, Adhar card holders based on their ID's. It can used in universities for sorting of student data such as marks. It can be used in sorting of financial data in banking sector.

REFERENCES

Anderson, T.E., D.E. Culler and D. Patterson, 1995. A case for NOW (Networks of Workstations). IEEE. Micro, 15: 54-64.

Boklund, A., S. Mankefors-Christiernin, C. Jiresjo and N. Namaki, 2005. COTS-cluster evolution during the last decade and extrapolation into the next. Proceedings of the 2005 Conference on Parallel and Distributed Computing and Networks (PDCN'05), February 15-17, 2005, ACTA Press, Calgary, Alberta, pp: 614-619.

Chhabra, A. and G. Singh, 2010. A Cluster Based Parallel Computing Framework (CBPCF) for performance evaluation of parallel applications. Intl. J. Comput. Theor. Eng., 2: 226-232.

Cirtek, P. and S. Racek, 2007. Performance comparison of distributed simulation using PVM and MPI. Proceedings of the International Conference on Computer as a Tool EUROCON 2007, September 9-12, 2007, IEEE, Warsaw, Poland, ISBN:978-1-4244-0812-2, pp: 2238-2241.

Ismail, M.A., S.H. Mirza and T. Altaf, 2011. Concurrent matrix multiplication on multi-core processors. Intl. J. Comput. Sci. Secur., 5: 208-220.

Khan, R.Z. and M.F. Ali, 2011. A comparative study on parallel programming tools in parallel distributed computing system: MPI and PVM. Proceedings of the 5th National Conference on INDIACom-2011, March 10-11, 2011, Bharati Vidyapeeth Institute of Engineering, Pune, India, ISBN:978-93-80544-00-7, pp: 1-7.

Kumar, R. and S. Jangra, 2015. Automated fault tolerance framework for alchemi desktop grid. Intl. J. Adv. Eng. Sci., 5: 11-20.

Kumar, R. and S. Jangra, 2016. Memory and CPU power based automatic fault tolerant framework in alchemi computational grid. Intl. J. Comput. Appl., 134: 13-18.

Kumar, S., R.K. Chauhan and P. Kumar, 2008. Minimum process error recovery algorithms or mobile distributed system using global checkpoint. Intl. J. Inf. Technol., 1: 25-33.

Meenal, C.D. and G.H. Prashant, 2014. Parallel programming models: A systematic survey. Intl. J. Comput. Sci. Inf. Technol., 5: 5268-5271.

Mostaccio, D., C. Dalforno, R. Suppi and E.L. Fadon, 2006. Distributed simulation of large-scale individual oriented models. J. Comput. Sci. Technol., 6: 59-65.

Sampath, S., B.S. Bharat and B.R. Nanjesh, 2013. Performance evaluation and comparison of message passing interface and parallel virtual machine. Intl. J. Adv. Res. Comput. Sci. Software Eng., 3: 200-206.

Sejwal, A., S. Jangra and Y.S. Sangwan, 2012. A decision support system for industry based upon multivariate spatial outlier detection techniques. Procedia Technol., 4: 401-405.

Shan, H. and J.P. Singh, 2001. A comparison of MPI, SHMEM and cache-coherent shared address space programming models on a tightly-coupled multiprocessors. Intl. J. Parallel Program., 29: 283-318.

Sharma, R.K., P. Kanungo and M. Chandwani, 2011. Performance evaluation of parallel applications using message passing interface in network of workstations of different computing powers. Indian J. Comput. Sci. Eng., 2: 184-187.

Sunderam, V.S., G.A. Geist, J. Dongarra and R. Manchek, 1994. The PVM concurrent computing system: Evolution, experiences and trends. Parallel Comput., 20: 531-545.