Volume 5, Issue 6, June 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Dynamic Load Balancing in Parallel Processing Using MPI Environment to Improve System Performance

Rajesh Tiwari
SSTC-SSGI Bhilai
(C.G.) India

Manisha Sharma
BIT Durg
(C.G.) India

Kamal K. Mehta
Nirma Univ. Ahmadabad
(Gujarat) India

Abstract*— Accuracy, efficiency and speed plays an important role in today computer world. This paper presents an algorithm, coded under message passing interfacing environment with some complex mathematical applications. Such applications were chosen with higher degree of similar iterations. Experimental setup has been completed on i5 processor computer. The Maximum number of concurrent process with minimum overhead is defined as break over point. Compiled results shows break over point, to stop further creation of concurrent processes. The proposed system will improve the speed of a system; the suggested algorithm divides the job into many cores , which run concurrently without affecting the performance of other applications.

Keywords*— MPI, Concurrency, Parallel models, SPMD, Vector processing, Load balancing, processes, Scatter, Gather, Reduce.

I. INTRODUCTION

Since last two decades, research initiatives had made it possible to use distributed architectures for high performance computing (HPC) [1]. HPC was done primarily on supercomputers. There are number of different classes of supercomputers, including vector, SIMD and MIMD. Parallel systems are now classified as: capability computing, the high-end computing architectures; cluster computing [2], parallel computers in a local area network built from commodity components; and grid computing [3] [4], a computation infra-structure designed on top of a wide area network such as the Internet. High-end computing architectures support nested parallelism levels and several memory hierarchies, making impossible, even the computer researchers, to exploit all their potential performance using general programming models. Clusters and grids are said to be the abstract that has been created new application for high performance computing, with increasing levels of complexity and scale. In consequence, there is no consensual model for high-performance computing that may be able to reconcile efficiency and portability with abstraction and generality in all application contexts. There is a enormous chances to achieve parallelism on stand alone machine using freeware tools available in the market. Some freeware tools available are Linux, MPI, PVM, CUDA(Compute Unified Device Architecture). These tools help researchers to convert a SISD program to SIMD program with lesser efforts. This will save the time and resources. Resources are the main component which shared by many programmers. Many programmer uses the same resource at a time, it will create a problem for the resource to whom it will reply, if all the jobs demanding to the resource at the same moment of time. Solution is proposed using Single Instruction Stream and Multiple Data Stream.

This paper presents a parallel model developed using MPI library for highly computational intensive applications. This work targets to have more precise results, to gain attention of high speed parallel algorithms developers. Input data sets applied in the range of 10,000 to 10,00,000 or more values of same attributes. Model had been developed for variety of computations, (like bubble sort and merge sort.

The paper is organized as follows: Section 2 represents the background, where load balancing, parallelism and MPI have been explained. Section 3 describes the program partitioning technique. Section 4 discusses the environment, parallel algorithm and the timing aspects. Experimental result is shown in Section 5. Section 6 states the conclusion and future work of the paper.

II. BACKGROUND

The SIMD programming model promotes a change of action in the practice of programming for high performance computing, by moving parallel programme from a process-based perspective to an orthogonal concern-oriented perspective. From the process-based perspective, a program is a collection of processes that interact though communication primitives**.** The level of abstraction for process-based parallel programme results loss in efficiency. The implementations of processes are found scattered, since concerns are orthogonal to processes. In fact, a process may be viewed as a collection of slices. Each slice will describe the role of the process with respect to a given concern. In this context, concern is the decomposition criterion for slicing processes [5].

A. Load Balancing Incorporation

Load balancing [6] is the major criterion to improve throughput and speedup of a running set of jobs while considering high processor utilization. Load balancing is the allocation of the workload among a set of available co-

operating processes [7]. Load balancing is very important for big applications where multiple processes works together, if load balancing is it not done properly then some process finished their jobs in less time and some process finished their jobs in more time. The process who finished their jobs in less time will be idle until the longer process will finished their jobs. Load balancing is used to minimize the idle time spent by any process. Using Load balancing technique all the processes finished their jobs in almost same time. Load balancing[16][17] is mainly of two types: Static Load balancing and Dynamic Load balancing. Dynamic Load balancing has been used to make sure that each host is doing its fair share of work, which can be a real performance enhancer. The partitioning of data occurs before the job starts, or as an early step in the application. The size and number of tasks can be varied depending on the processing power of a given machine. On a lightly loaded network, this scheme can be quite effective. When computational loads vary, a more sophisticated dynamic balancing method is required. The most popular method is called the Centralized Work Pool method. This is typically implemented as a Master/Slave [8] program where the master manages a set of tasks (queue). It sends jobs to slaves to do as they become idle. This method is used in the sample program supplied with the distribution. This method is not suitable for applications requiring task to task communication, since tasks will start and stop at arbitrary times. Computation terminates when the task queue is empty and every process has made a request for another task without any new tasks being generated. Variations in load balancing are possible for different applications.

B. Programming Paradigm

Sorting applications are suited to functional parallelism, others to data parallelism. In functional parallelism, different machines do different tasks based on their specific capabilities. For example, a supercomputer may solve part of a problem suited to vectorization, a multiprocessor may solve another part suited to parallelization, and a graphics workstation may be used to visualize the data in real time. In data parallelism, the data is distributed to all of the tasks in the virtual machine. Operations (often quite similar) are then performed on each set of data and information is passed between processes until the problem is solved. Program codes were written to run all modules in parallel way. Here Instruction level parallelism (overlap among instruction is called ILP) is used.

According to Flynn's classification Single Instruction Stream and Multiple Data Stream (SIMD) is used in Load Balancing. Single instruction is applied to complete set of data and programming is based on Single Program Multiple Data (SPMD) [9] [10][15] concept. Basic reason to use SPMD is to use existing hardware available in laboratory. SPMD model for parallel programming is supported by standard communication libraries of MPI [11] [12] parallel programming paradigm.

supercomputer may solve part of a problem suited to vectorization, a multiprocessor may solve another part suited to parallelization, and a graphics workstation may be used to visualize the data in real time. In data parallelism, the data is distributed to all of the tasks in the virtual machine. Operations (often quite similar) are then performed on each set of data and information is passed between processes until the problem is solved. Program codes were written to run all modules in parallel way. Here Instruction level parallelism (overlap among instruction is called ILP) is used.

According to Flynn's classification Single Instruction Stream and Multiple Data Stream (SIMD) is used in Load Balancing. Single instruction is applied to complete set of data and programming is based on Single Program Multiple Data (SPMD) [9] [10][15] concept. Basic reason to use SPMD is to use existing hardware available in laboratory. SPMD model for parallel programming is supported by standard communication libraries of MPI [11] [12] parallel programming paradigm

C. Message Passing Interface (MPI)

MPI is a library which is used for parallel programming in association with C, C++. It is a research implementation of the MPI-1.2 standard [12] with use of advance functions of MPI-2 standard. In doing so, various modules that tune MPI's runtime functionality are available to the programmer, including TCP and shared-memory [13] communication. However, this work is directly applicable to any MPI implementation, as the replication [14] work itself is neutral implementation. Message Passing Interface is a specification for message passing libraries, designed to be a standard for distributed memory, message passing and parallel computing. The goal of the Message Passing Interface is to develop a widely used standard for writing message-passing programs. The interface should establish a practical, portable, efficient, and flexible standard for message passing.

Every process has its own unique, integer identifier (rank) assigned by the system when the process initializes. A rank is sometimes also called a "process ID". Ranks are contiguous and begin at zero.

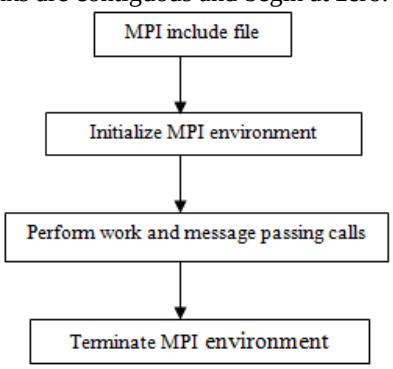

Figure 1. Structure of MPI Program

A group is an ordered set of "N" processes. Each process in a group is associated with a unique integer rank. Rank begins at 0 and range in sequence to "N-1". A group possesses its own unique identifier assigned by the system and it's unknown to the user. That is, a group is represented by an opaque object - the user does not know the internal structure and must call an inquiry routine to determine its attributes, The structure of the MPI Program is given in Figure 1.

.

III. PROGRAMING PARTITIONING

Some problems can be solved in parallel as follows. First decompose the problem of size N into two or more smaller sub parts. Suppose that the required decomposition, when done in parallel, takes $Td_{(n)}$ time. Partitioning is done in such a way that all problems are solved independently to obtain corresponding results. As the sub problems are smaller than the original one, the time Ts to solve them will likely be less than $T_{(n)}$. Finally, combine the results of the sub problems to compute the answer to the original problem. If the time taken to combine the result takes $Tc_{(n)}$, then the total computation time is expressed as

$$T_{(n)} = Td_{(n)} + Ts + Tc_{(n)} \quad \text{..... [1]}$$

IV. EXPERIMENTAL WORK

This section explains the computing environment, parallel algorithm developed and timing observations.

A. Computing Environment

Fundamental objective of experimental work carried out is to use easily available resources generally marked as too conventional. Hardware used for parallel model is Core 2 Duo (having parallel processors) , working with 2.2 GHz frequency,1 GB RAM, 32bit environment operating system (Fedora 11/Red hat 5.0 or higher), C with MPI Support.

B. Parallel Algorithm

The main purpose of this section is to show a parallel algorithm ,which is given as follows:
Step 1. Generate N random numbers.
Step 2. Start the clock to record the starting time.
Step 3. Divide (Scatter) the N data equally to m Processes.
Step 4. All the processes sort the data independently.
Step 5. Balance the load b/w the processes.
Step 6. Collect (Gather) the data from m processes and arrange them .
Step 7. Stop the clock.
Step 8. Calculate the lapsed time.
Step 9. Store the sorted data in a file to check whether the input data is sorted or not.

C. Timing Aspects

Proposed parallel model SIMD requires comparison with sequential model SISD . It gives rise to include timing aspects in parallel program development. The time required by major tasks are :

1. Generation of input data for parallel operation.
2. Parallel Pre-processing (scheduling, load balancing).
3. Actual task performance (specific to selected application).
4. Generation of output.

Parallel processing requires huge amount of data, so 10,00,000 input data have been generated using the random function available in 'C' language. The time taken to generate the number is not considered because it is operational overhead and more towards setting up the environment. In this method huge volume of input data had been taken and memory is allocated dynamically. The algorithm has been developed in such a way that the timer starts, after generation of all input numbers. The "starttime()" returns the current time of the system clock . After completion of sorting "stoptime()" has been used which returns the current time of the system clock. We calculate the total elapsed time during this period as:

$$Elapsed\ time = \frac{(starttime\ (\ ) - stoptime\ (\ ))}{CLOCK\_PER\_SEC} \quad \text{.....[2]}$$

Where CLOCKS_PER_SEC is system specific parameter. Command line argument have been used to specify the number of process for every execution. Load balancing expression has been coded carefully to have symmetric load distribution to all the processes.

Sorted data has been collected in a file. The time needed to transfer data into file has not been included in the sorting time in this paper.

V. EXPERIMENTAL OUTPUT

Two different applications Merge/Bubble sort had been selected to write parallel algorithm.
Table 1 shows the time required to sort the data using Merge sort with varying number of processes ranging from 1 to 128. Figure 2 shows that the Merge sort reduces the sorting time. When two numbers of processes had been used the sorting time was 0.32 sec. As the number of processes increased the sorting time decreased. When the 64 number of processes used the sorting time had been 0.09 sec. After that as the number of processes increased the sorting time also increased.

Table 1.Observed values for Merge sort

No. of Process	Merge Sort time (in sec.)
1	1.01
2	0.32
4	0.2
8	0.17
16	0.16
32	0.13
64	0.09
128	0.47

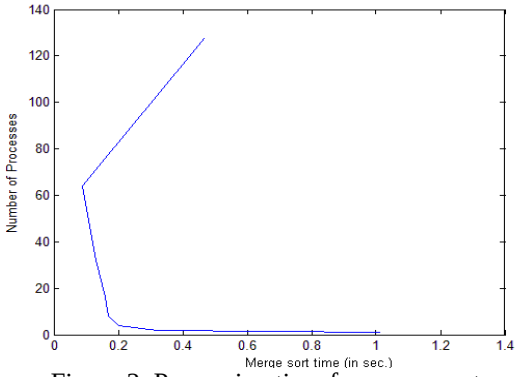

Figure 2. Processing time for merge sort

Table 2 shows the time required to sort the data using Bubble sort with varying number of processes .ranging from 1 to 128. Figure 3 shows that the Bubble sort reduces the sorting time. When two numbers of processes had been used the sorting time was 1583.46 sec. As the number of processes increased the sorting time decreased. When the 64 number of processes used the sorting time has been 2.16 sec. Bubble sort work properly up-to 128 processes. After that as the number of processes increased the sorting time also increased.

Table 2 . Observed values for Bubble sort

No. of Process	Bubble Sort time (in sec.)
1	1724.83
2	1583.46
4	394.7
8	98.57
16	24.8
32	6.23
64	2.16
128	0.81

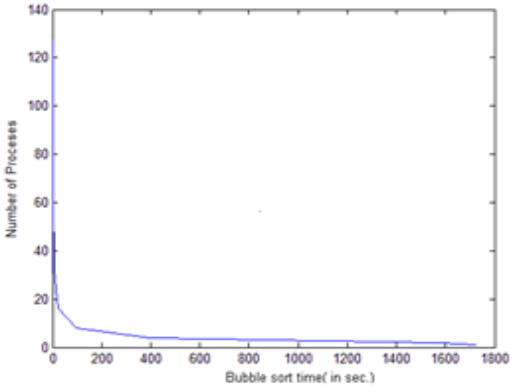

Figure 3. Processing time for bubble sort

Comparative analysis has been based on result timing observed from experiment. Interpretation shown in Figure 4 indicates that initial response of bubble sort is not appreciable. But after 32 processes the sorting time rapidly decreased till 128 number of processes. After that the sorting time has been increased. Similarly the merge sort spends less sorting time till 64 processes. The algorithm gave better result for merge sort as well as bubble sort for 64 number of processes.

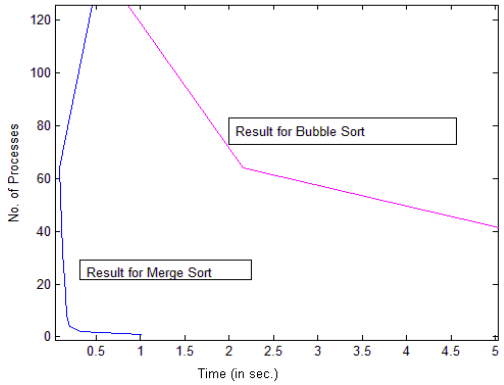

Figure 4. Comparative analysis of processing time

VI. CONCLUSION & FUTURE WORKE

Experimental work has been carried out on standalone machine with dual core architecture. Communication latency has supposed to be negligible. As result states that, conventional set up can also be used with freeware softwares as MPI, openMP, Linux etc. It is highly cost effective and only effort needed to write parallel logic. Result says that the algorithm in this paper is more suitable for Merge sort as compare to Bubble sort. Sorting 10,00,000 data ,required only 0.09 sec. for Merge sort and 0.81 sec. for Bubble sort. This method will work in a more efficient way, if the input data increases up-to 1,00,00,000 random data's.

Further expansion of this work would be to perform the same experiment on Quad core architecture and to set up parallel model in computer network environment.

REFERENCES

[1] Gerd Heber , David Lifka and Paul Stodghill , "Post – Cluster computing and the Next Generation of Specific Applications" Research is partially supported by NSF grants EIA-9726388,EIA-9972853 and ACIR -0085969.

[2] R. Buyya (ed.). "High Performance Cluster Computing: Architectures and Systems" Prentice Hall, First Edition.

[3] I. Foster and C. Kesselman. "The Grid 2: Blueprint for a New Computing Infrastructure" M. Kauffman, 2004.

[4] Sanjay Patel and Madhuri Bhavsar, " QoS Based User Driven Scheduler For Grid Environment " , Advanced Computing : An International journal , Vol. 2 ,No. 1 pp 18 – 26, 2011.

[5] F. Tip. "A Survey of Program Slicing Techniques" , Journal of Programming Languages, Vol. 3, pp 121–189, 1995.

[6] Ka-Po Chow and Yu-Kwong Kwok , "On Load Balancing for Distributed Multiagent Computing", IEEE Transactions on Parallel and Distributed Systems , Vol. 13 , No. 8 , pp 787 – 801 , 2002.

[7] Armando E., Marcelo R., Naiouf, Laura C., "Dynamic Load Balancing in Parallel Processing on Non – Homogeneous Clusters", Journal of Computer Science , Vol. 5, No. 4 , pp. 272-278 ,2005.

[8] S. Jeyakumar and S. Sundaravadivelu , "Parallel Adaptive Motion Estimation With Dynamic Load Balancing For Fast Video Compression" , Journal of Advance Research in Computer Engineering: An international journal, Vol. 3, No.2, pp 219-227, 2009 .

[9] E. Smirni and E. Rosti ," Modelling Speedup of SPMD Applications on the Intel Paragon : A Case Study,", Proc. Int'l Conf. and Exhibition High- Performance Computing and Networks , Languages and Computer Architecture 95, pp. 94-101, 1995.

[10] Rajesh Tiwari, Rohit Raja, Nishant Behar Kamal Mehta, "Concurrent Solution of SPMD problem USING MATLAB", Journal of Advance Research in Computer Engineering: An International Journal, Vol. 3, No.2, pp 209-212, July – Dec. 2009 , ISSN – 0974 - 4320..

[11] G. Burns, R. Daoud, and J. Vaigl, "LAM: An Open Cluster Environment for MPI", Proc. Supercomputing Symp., pp. 379-386,1994.

[12] The MPI Forum, "MPI: A Message Passing Interface", Proc. Ann. Supercomputing Conf. (SC '93), pp. 878-883, 1993.

[13] Radenski , A. , " Shared Memory , Message Passing and Hybrid Merge Sorts for Standalone and Clustered SMPs", Proc. PDPAT' 11 , the 2011 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications , CSREA Press , pp 367 – 373, 2011.

[14] John Paul Walters , Vipin Chaudhary , " Replication-Based Fault Tolerance for MPI Applications " , IEEE Transactions on Parallel and Distributed Systems , Vol. 20 , No. 7 , pp 997 – 1010, 2009.

[15] Rajesh Tiwari et al., "An Improved Approach for Concurrent Solution of SPMD Problem", CSVTU Journal of Engineering, Bhilai (C.G.) Vol. 5, 2012, pp 70-74, ISSN 0974 - 8725.

[16] Rajesh Tiwari et al. , " The Efficient load balancing in the parallel Computer" , International Journal of Advanced Research in Computer and Communication Engineering , Vol. 2, Issue 4, April 2013 , pp 1667 – 1671 , ISSN : 2319-5940

[17] Rajesh Tiwari et al. , "Analysis of Various Decentralized Load Balancing Techniques" International Journals of Recent and Innovative Trends in Computing and Communication (IJRITCC), Vol2 , Issue 11 , November 2014, pp 3360 – 3365 , ISSN: 2321 – 8169.

