3rd National Conference on Green Technology and Science for Sustainable Development
© 2020 IJSRST | Volume 5 | Issue 6 | Print ISSN: 2395-6011 | Online ISSN: 2395-602X
International Journal of Scientific Research in Science and Technology

Implementing Load Balancing in Parallel Computing through Program Slicing

Dr. P. A. Tijare
Department of Computer Science and Engineering, Sipna College of Engineering & Technology, Amravati, Maharashtra, India

ABSTRACT

We need to achieve more efficiency in execution time in case of problem solving using the scheduling techniques. Parallel computing shows considerable potential to deliver cost effective solution to application with high requirements in performance. The proposed work potentially reduces the time and effort needed to develop large variety of parallel application. Load balancing in parallel computing is achieved through slicing the iterative program and running the slice on the other nodes to improve the performance of the system.

Keywords : Load Balancing, Slicing, Parallel Computing

I. INTRODUCTION

Parallel Computing is a form of computation where computer problems are divided into smaller jobs and then execute in parallel. There are different forms of parallel computing: Bit-Level Parallelism, Instruction Level Parallelism, Data Parallelism and Task Parallelism. Parallelism has been employed for many years, mainly in High- Performance Computing, but due to physical constraints the frequency scaling is not achieved till date.

The recent progress in the parallel computing has provided many facilities in transmission and manipulation of data through network. However, these facilities and advancement have also brought the challenges in the field of parallel computing. High Complexity of building parallel application is often cited as one of the major impediments to the mainstream adoption of Parallel Computing. Many Scientific Computations require a large amount of computing time, the computing time can be reduced by dividing a problem over several processors.

II. RELATED WORK

Message Passing Interface is the most important parallel programming tool in cluster computing currently [1]. Message passing achieves the communication of parallel program by adopting message passing mechanism. MPI has many merits such as good portability, strong performance, and high efficiency and so on, and it has all kinds of free, practical versions, for example MPICH, LAM, IBM MPL, all of parallel computer companies support MPI, Achieving load balance become very interesting in parallel program, it can improve the performance of MPI program.

Message Passing Interface is a typical interface of message passing mechanism. It can exploit parallel program based on message passing. Message Passing Interface adopts program model of signal process signal data, in other words, each process carries out the same message passing interface program.

In the Research Paper "Dynamic Load Balancing Algorithm for MPI Parallel Computing" by Sun Nian, Liang Guangmin they proposed an implementing method in Message Passing Interface parallel program that can transfer tasks between nodes effectively by finding node's load.

The Problem of Dynamic Load Balancing need to be solved in order to make cluster system high performance. Load balancing problems of nodes are main barrier, being one of hotpots of scientific research domain. Load balance includes static load balance and dynamic load balance, static load balance is not relate to state of system and it has low efficiency; Dynamic load balancing algorithm can balance the loads of nodes, having good practicability [2].

"A Generic Parallel Computing Model for Distributed the Environment" by Chi-Chang Meng-Xiang Chen, proposed a generic parallel computing model for distributed computing platforms. They have provided a generic model for the users who intent to write parallel programs over the distributed system with CORBA middle ware [5]. They used the concept of factory object in the model to create slave object dynamically to fulfill the master/slave parallel computing paradigm [4].

Cluster computing technology is increasingly becoming the technology of choice for the parallel computing in recent years. This is mainly due to the low cost and easy expansion of cluster workstations in comparison to the main frame parallel computer systems. CORBA (Common Object Request Broker Architecture) is one of the most popular middle wares for the heterogeneous distributed system supported by OMG [3].

The author of the research paper "Design Patterns for Parallel Computing Using a Network of Processors", Stephen Siu and Ajit Singh provides a DPnDP model and system that aims to provide pattern in the form of parameterized and application independent library of code skeleton[6].

The Design Pattern and Distributed Processes uses Multiple Instruction Multiple Data (MIMD) Processor Architecture and Operating system which creates process and pass message for communication. In their model, Parallel Program is represented by a directed graph. Each node in the graph has a set of input and output ports that are used for receiving and sending messages respectively. Output port of a node is connected to an input port of another node.

When a node sends a message to one of its output ports it reaches the input ports of the connected node which can receive this message. They have found the abstraction of nodes and ports quite valuable in designing and using our design-pattern based system. It has allowed the model to remain independent of the specifics of the underlying message passing models such as Sockets [8], PVM [7] or MPI [9].

III. PROGRAM SLICING

A computer program is an accumulation of directions given to Computer for performing particular task. A computer obliges projects to work, and other programming ideal models to execute the program's instructions in a central processing unit.

A. Program Flow

Each computer program has a begin and end point and exceptional standards how the processor will execute every instruction. It fluctuates from Program to Program. Generally, a program will begin at some "main" function and proceed "downward" or "upwards" one line at any given moment until the finish of the capacity is come to. In the event that the conditional statement occurs in the program the program control will stream to the following explanation for execution relying on the condition.

In the event that a loop is occurred in a program, a block of statements can be executed at least zero or

more time depending upon the condition. On the off chance that the "Control Statement" is absent in the program, the program will dependably spill out of line to line.

B. Data Flow

Data flows through a program in proper sequence. The variable in the statement is changed in each operation and put the new value in new variable. Each line in the program executed autonomously or consecutively. Every announcement takes already computed data, changes it and put away in another variable.

Information or data is made, put away in a specific variable, and afterward used to figure new information, which is similarly put away in new variable and replaces the old values from old variables.

C. Program Representation

To have intermediate representation of a program is the best way to understand large programs. To compute a slice, it is first required to change the given program source code into a particular intermediate representation. A general feature for most of the slicing algorithms is that programs are represented by a directed graph.

There are many graphical representations such as:

1. Control Flow Graph
2. Data Dependence Graph
3. Control Dependence Graph
4. Program Dependence Graph
5. System Dependence Graph
6. Extended System Dependence Graph, etc.

D. Control Flow Graph (CFG)

A control Flow Graph represents all paths, using graph notations that might be traversed through a program during its execution. In a control flow graph each node in the graph represents a basic block. Directed edges are used to represent jumps in the control flow [10,11].

A CFG contains a node for each statement and control predicate in the program; an edge from node i to node j indicates the possible flow of control from the former to the latter. CFGs contain special nodes labeled START and STOP corresponding to the beginning and the end of the program, respectively.

E. Slicing

Program slicing is a method of creating slices of the program according to slicing criteria. Different approaches on Program Slicing of different researches are there for different types of program. Our main focus was basically making the real use of slicing algorithm in the computing world. The real use of slicing is only when the slices are independent in all respect i.e. running independently by taking the necessary input, reliability with respect to necessary output. Traditional program slicing is done specifically with slicing criteria. The Slicing algorithm developed in research work use slicing point as backbone of the algorithm. A proposed Slicing Algorithm is designed only for Iterative Program.

The Slicing Point in the program is the Point where the logic of the program repeats the number of the time till the logic of the program is complete. In this research work we refer the slicing point as “For Loop”. The Iterative Program contains repetition of the same logic number of times, and for the completion of the repetition, the system waits till the test condition is completed. The Slicing Point is the point from where we can slice the program into number of required slice.

F. Slicing Algorithm Steps

The Slicing Algorithm is described in the following Steps:

1. Formed CFG of the Program.
2. Identify Slicing point of the Program.
3. Find initial and last value of the Iteration in slicing Point.
4. From Last Value of the iteration divide the Slicing Point in number of iteration.

5. In newly formed slice, add the program starting statement and the statement after the slicing point to the slice, so that the slice can be ready for execution.

CFG of a program before slicing

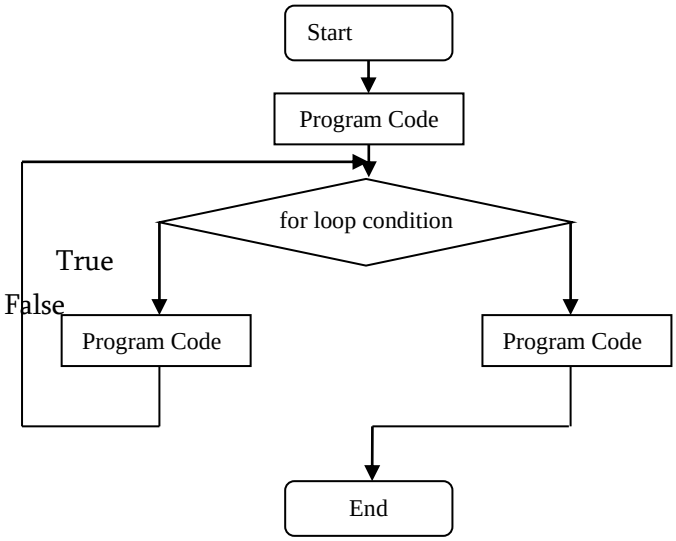

Figure 1 : CFG of a program before slicing

CFG of a program after slicing

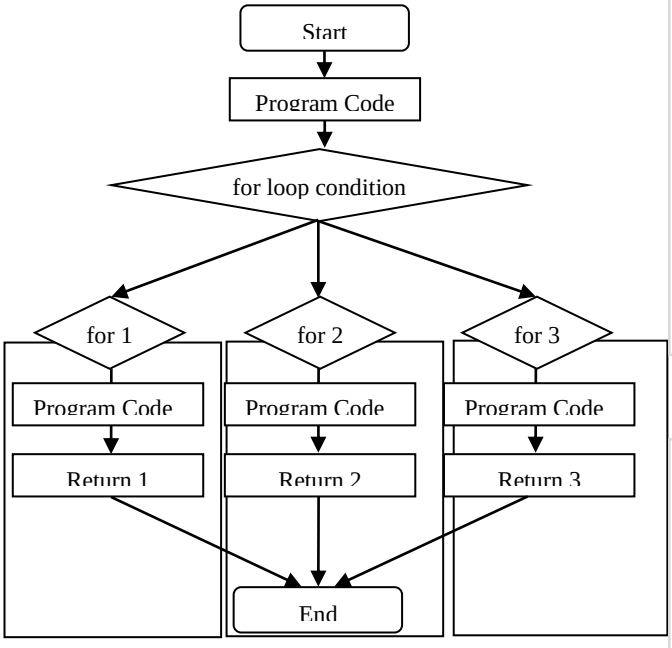

Figure 2 : CFG of a program after slicing

Figure 1 shows the Control Flow Graph of a program and figure 2 shows the Control Flow Graph of the program after applying the Slicing Algorithm to the program.

IV. Results

Here we considered three numbers of systems present in the network. The task is sliced and sent to number of nodes for partial execution. Initially task is completed on local system.

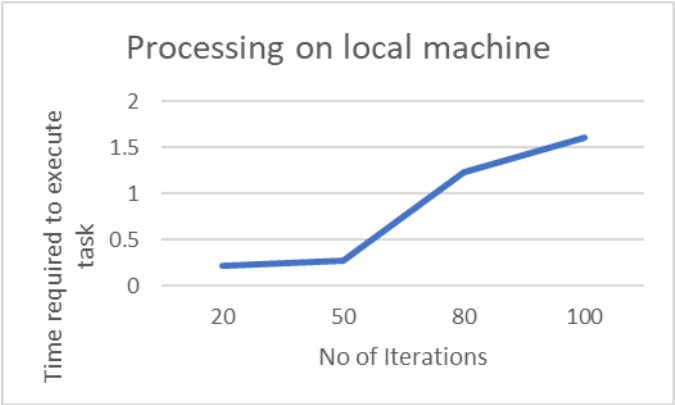

Figure 3 : Processing on local machine

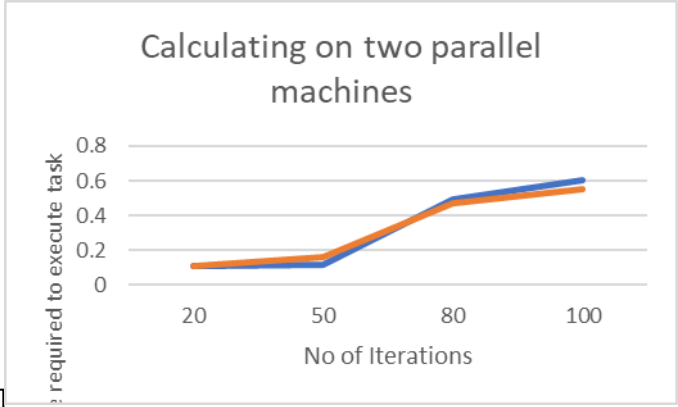

Figure 4 : Calculating on two parallel machines

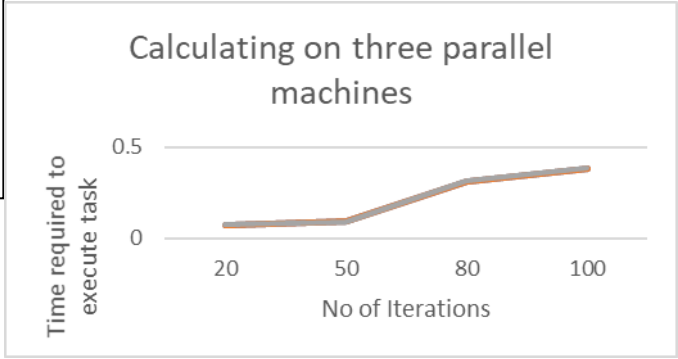

Figure 5 : Calculating on three parallel machines.

V. CONCLUSION

Implementing execution of a task in parallel that can run for number of iterations between nodes effectively by consideration of node's load is carried out. Reduction in completion time of a task is proved by the execution of the algorithm in parallel computing. Thus we have achieved Load balancing in parallel computing through slicing the iterative program and running the slice on other nodes to improve the performance of the system.

VI. REFERENCES

[1]. Zhu Yongzhi, Zhao Yan, Wei Ronghui, "Constructing and Performance Analysis of a Beowulf Parallel Computing System Based on MPICH", Computer Engineering and Application 2006.14:132.

[2]. Sun Nian, Liang Guangmin "Dynamic Load Balancing Algorithm for MPI Parallel Computing", 2009 International Conference on New Trends in Information and Service Science.

[3]. Zambonelli.F, "Exploiting Biased Load Information in Direct neighbour Load Balancing Policies J]", Parallel Computing, 1999, 25(6): 745-766.

[4]. Chi-Chang Chen Meng-Xiang Chen, "A Generic Parallel Computing Model for the Distributed Environment", 2006, IEEE Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies.

[5]. Gerald Brose, Andreas Vogel, and Keith Duddy," Java Programming with CORBA". Third Edition, John Wiley & Sons, 2001.

[6]. Stephen Siu and Ajit Singh, "Design Patterns for Parallel Computing Using a Network of Processors", 1997, Sixth IEEE International Symposium on High Performance Distributed Computing.

[7]. G. Geist and V. Sunderam. Network-based concurrent computing on the PVM system. Concurrency: Practice and Experience, 4(4):29:3-311, 1992.

[8]. S. Leffler, M. McKusick, M. Karels, and J. Quarterman. The design and implementation of 4.3 BSD UNIX Operating System. Addison-Wesley Publishing Company, Inc., 1990.

[9]. D. Walker. The design of a standard message passing interface for distributed memory concurrent computers. Parallel Computing, 20(4):657-673, 1994.

[10]. F. Tip. 1995. A survey of program slicing techniques. Journal of Programming Languages 3(3):121–89.

[11]. J. Ferrante, K. Ottenstein, and J. Warren. 1987. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems 9(3):319–49.