Simulation Study for Performance and Prediction of Parallel Computers

Neeraj Kumar

Submitted in January 2012; Accepted in June 2012

Abstract - ***The issue of performance evaluation and prediction has concerned the users throughout the history of computer evolution. In recent times the parallel computer is gaining popularity as an effective solution to low cost supercomputing. In this study we discuss a simulation study, performed for evaluating the performance of parallel computers connected in different topologies.***

Index Terms - ***Performance measures, Processor utilization, System utilization, Throughput***

1. INTRODUCTION

The future need of much more powerful super computation asks for parallel (digital) computers, containing a large number of fast processors that can cooperate quickly and efficiently. In parallel processing, high performance data processing and data flow are of equal importance. In practice so far, loss of efficiency often happens for the technical reason that the communication system of a parallel computer has not enough capacity. Lack of communication capacity will result in transfer bound processing instead of computation bound processing. Loss of efficiency also often happens because a parallel algorithm is still in the early stage of development. That makes it difficult to define the architecture and programming of a parallel computers such that, efficient implementation of parallel algorithms is possible in a wide range of applications. Moreover, the applicability of parallel computation is hampered, since the programming in parallel computation is still more difficult than programming in serial computers.

The need for computer performance evaluation exists from the initial conception of a system's architectural design to its daily operation after installation. In the early planning phase of a new computer system product, the manufacturer usually makes two types of predictions. The first type is to forecast the nature of applications and the levels of system workloads of these applications. Here, the term workload means the amount of service requirements placed on the system. The second type of prediction is concerned with the choice between architectural design alternatives, based on hardware and software technologies that will be available in the design period of the planned system. Here the criterion of selection is known as cost performance trade off. The accuracy of such prediction rests, to a considerable extent on the capability of mapping the performance characteristics. Such translation procedures are by no means straightforward or well-established. After the architectural decisions have been made and the system design and implementation started, the scope of performance evaluation becomes more specific. The interactions among the operating system components—algorithms for job scheduling, processor scheduling, and storage management must be dealt with, and their effects on the performance must be predicted. Comparing the predicted performance with achieved performance often reveals major defects in the design or errors in the system programming. Now, it is universally accepted that the performance evaluation and prediction process should be an integral part of the development efforts, throughout the design and implementation activities.

Special Centre for Sanskrit Studies, Jawaharlal Nehru University, Delhi, INDIA
E-mail: neerajdagar685@gmail.com

2. MEASURES OF PERFORMANCE

When it is said that the performance of the computer is great, it means, perhaps, that the quality of service delivered by the system exceeds the expectation. But the measure of service quality and the extent of expectations vary depending on the individuals involved, eg, system designers, installation managers, terminal users, etc. If an attempt is made to measure the quality of computer performance in the broadest context, then issues like user response (as well as the system response), ease of use, reliability, user's productivity, etc must be considered as the integral parts of the system's performance. Since the performance analysis cannot avoid issues that are ultimately behavioural, the scope of this is discussed only in terms of clearly measurable quantities. This is done in the conventional way as, for instance, the signal-to-noise ratio probability of decoding errors as measures of performance of communication systems.

The performance measures can be classified into two broad categories:

(i) user oriented measures, and
(ii) system oriented measures.

The user oriented measures include such quantities as the turnaround time in a batch system environment and the response time in a real time and/or interactive environment.

The turnaround time is the length of time that elapses from the submission of the job, until the availability of its processed result. In the similar way, in an interactive environment, the response time of a request, represents the interval that elapses from the arrival of the request until its completion in the system.

Usually jobs are categorized according to their priority classes. Many factors may determine the assignment of priority to a job: the job's urgency, its importance and its resource demand characteristics and utilization.

Throughput is defined as the average number of jobs processed per unit time. It provides the degree of productivity that the system can provide. But in this case, throughput is not an adequate measure of performance; rather it is a measure of system workload.

2.1 System Utilization

In an execution cycle, all the processors may not participate in execution and may be idle throughout an execution cycle, waiting for results from other processors. The utilization of the system in terms of the number of processors used in an execution cycle is quantified by the parameter S_u, which is referred to as system utilization.

An algorithm has been considered which is executed in r cycle on P processors. Suppose, in an execution cycle of t_1 time units, P_1 processors are used, and in the next execution cycle of t_2 time units, p_2 processor are used , and so on then,

$S_u = (P_1*t_1+P_2*t_2+\ldots\ldots+P_r*t_r)/(P*(t_1+t_2+\ldots\ldots+t_r))$.

2.2 Processor Utilization

When the sub-domains assigned to different processors are not equal, then some processors finish computation earlier than others. As synchronization takes place at the end of every cycle, these processors wait for others to finish. This leads to idling and under- utilization of some processors which is quantified by the parameter P_iu for processor i. It characterises the load balancing of the system. Perfect load balancing occurs when the sizes of the sub-domains assigned to all the processors are equal, i.e, when $P_iu=1$, for i=1,2,….,p (where P is the number of processors in the system).

2.3 Inter-Processor Communication Time

In a message passing through multiprocessor, if $t_{start-up}$ represents the message start-up overhead or latency; t_{send} represents transmission time (which is inverse of the link bandwidth); 'k' bytes between two neighbouring processor involve a communication time, $t_{comm}=t_{start-up}+t_{send}*k$.

When the communication is not between two near neighbours, the communication time is estimated by assuming that it takes place in hops, and each hop corresponds to a near neighbour communication. The communication time between two processors is $n*t_{comm}$, where n is the number of hops by which the two processors are separated.

3. ANALYSIS OF PARALLEL ALGORITHMS

Once an algorithm for a new problem has been developed, it is usually evaluated using the following criteria: running time, number of processor used and cost[1].Besides these standard metrics, a number of other technology related measures are sometimes used when it is known that the algorithm is destined to run on a computer based on that particular technology.

Running Time

As the speed is emerging to be the main reason behind the growing interest in the field of parallel computers, the most important measure of a parallel algorithm is, therefore, the running time. According to AK1[1], running time is defined as parallel computer, that is, the time elapsed from the moment the algorithm starts to the moment it terminates. If the various processors do not begin and end their computation simultaneously, then the running time is equal to the time elapsed between the moment the first processor to begin computing starts and the moment the last processor to end computing terminates.

In evaluating a parallel algorithm for a given problem, it is quite natural to do it in terms of the best available sequential algorithm for that problem. Thus a good indication of the quality of a parallel algorithm is the 'speed-up' it produces.

This is defined as

Speed-up= (worst-case running time of fastest known sequential algorithm for the problem)/(worst-case running time for the parallel algorithm).

3.1 Number Of Processors

The second most important criterion in evaluating a parallel algorithm is the number of processor it requires to solve a problem. It costs money to purchase, maintain and run computers. When several processors are present, the problem of maintenance, in particular, is compounded, and the price paid to guarantee a high degree of reliability rises sharply. Therefore, the large the number of processor an algorithm uses to solve a problem, the more expensive it becomes to obtain the solution. For a problem of size n, the number of processors required by an algorithm, a function of n, will be denoted by p(n). Sometimes the number of processor is a constant independent of n.

4. IMPLEMENTATION

In traditional implementation of parallel programs, there is often no way of ensuring that the code implements designer's intentions. For example, a simple typographical mistake during coding can cause two processor to communicate when they should not, leading to disastrous, unpredictable consequences. If the design specifications could somehow be fed directly to the language processor, this unintended communication could be diagnosed syntactically. Inorder to be viable, the design must be formally defined as a computer language.

5. DESIGN OF SIMULATOR

In this simulator, a multiprocessor environment is simulated to evaluate the performance of different standard computation under various topologies. All the standard topologies like bus, ring, torus, hypercube, mesh, and tree are considered.

The simulation is done in c language

5.1 Assumptions

The model proposed here for performance prediction assumes that all inter processor communication times can be estimated a priori and that there are no unpredictable queuing delays in the system. An input file, having two fields containing

processor-ID name and process, and also the communication file is available. It is also assumed that any process can complete its message passing in one communication cycle if the route is free and the receiving process is ready.

5.2 Model

The input to this simulator is given after balancing load with a suitable load balancing technique. Here at each processor two queues are maintained: a ready queue, and a communication queue. In the beginning the ready queue at each processor contains all the processes assigned to that processor and the communication queue is kept empty. The round_ robin job scheduling technique is followed at each processor, ie, each process at a processor, is given a time slice for execution. An execution cycle is followed by a communication cycle. In the processes requiring communication among themselves communicate. Before any of the two processes communicate, first the links connecting them through the shortest path are examined. Then the communication queue of the partner processor is searched for the partner process. If it is found there, the communication delay is added to the respective counters and the partner process is removed from the front of the ready queue and is placed at the rear of the ready queue. When all the queues are exhausted then the program terminates. Computation time is added to each process at the end of each computation cycle. It also calculates time of completion of each queue. That is done by adding execution time of all the processes at each processor separately.

The different parameters and structures are described as follows.

Structure processor includes

(i) Current state of processor, ie, 'o' for every 1 for ready and 2 for idle.
(ii) Time_ stamp, clock, link clock for each link; and
(iii) Three process queues. Each queue has its own count.
  (a) Ready_ queue of active processes waiting for communication.
  (b) Communication_ queue of inactive processes waiting for communication.
  (c) Wait queue of inactive processes waiting to be creates as threads.

Proc_ array is dynamically allocates array of processors.

The declaration for the above is made as follows.

```
Struct processor {
Int current_ state;
Unsigned double time_ stamp , clock,*link_clk;
Int ready_process_count,comm_process_count,
Wait_process,count;
Struct process*ready_q_tl,*wait_q_t,(* comm._q_tl;
        }**proc_arr;
```

Structure process includes

(i) Process_ id identification of the process;
(ii) Priority of the process : 1 if urgent else 0;
(iii) Current state of process 0 if over and 1 if ready;
(iv) Partner_ proc : communication partner processor

Partner_ process: communication partner Process;

(v) Instruction queue and instruction count; and
(vi) Pointer to the 'next' process in the linked list.

The structure process is defined as follows.

```
Struct process {
Int priority, process id, inst_ count state; current state;
Int partner_ proc, partner_ process;
Unsigned double clock;
Struct process*next;
Struct inst_list,*instr_ hd, *instr_ tl;
};
```

The structure instruction list includes

(i) Type: integer value indicating the type of instruction;
(ii) Params array : parameter required for that instruction; and
(iii) Pointer to the next instruction in the linked list.

The declaration for list_ list is given as follows:

```
        Struct instr_ list {
        Int type;
        Int params4;
        Struct instr_ list*next;
        };
```

Other variables declared include t_ calc which store the total computation time, ie, the time required to run the same application on a single processor.

Initialize ( )

The initializing subroutine is a semi- interactive subroutine which initializes all parameters used afterwards by the simulator. Here the number of nodes/ processors type of topology

Processor used and its frequency are taken as input. Also the clock used is initialized. All the process counts are also initialised.

get_link(int sp.int dp)

This subroutine takes the destination and the source processor as input (as well as topology)and returns communication link between those two processors. Here popular topologies like mesh, star, hypercube, tree, torus and wk_recursive are consider as well as the logic topologies like ring, pipeline, etc.

read_input(cha*filename)

This function reads the input file given in command line argument. Here declaration of various dummy statements is given which is the output file of the parser. Parser replaces the actual parallel C statements by these dummy Statements considering the worst case of execution. Here the queues for different processors are maintained to be used by the simulator. There are several smaller procedures doing different tasks.

void create(int*pro_arr)
This creates another (thread) process. This thread which was initially stored in wait queue and is moved to read queue. Delay is added to the process and processor clock, and the process input in the ready queue end.
void send(int*par_arr)
This performs the communication operation 'send'. At first the partner processor of processes is updated. Then the communication queue of the partner processor is reached to find out whether it contains partner process or not. If it is able to find partner process, then corresponding communication delay is added to both the processes. The partner processes is removed from the communication queue of the partner processor or is put in its ready queue, and the ready queue and communication queue of the partner process are updated. On the other hand, if it is unable to find partner process in the communication queue of the partner processor then the current process is put in the communication queue of the current processor. If it is able to find the partner processor, corresponding link delays are added, and the current process is put in the ready queue end.
void receive(int*pro_arr)
This function performs the communication operation 'receive'. Here, first the communication queue of the partner processor is reached for the partner process. If it is found, then communication delay is added to both the processes. The partner process is removed from the communication queue of partner processor or is put in its ready queue. The communication queue and ready queue of partner process are updated. If unable to find the partner process in communication queue of partner processor then the current process is put in the communication queue of the current processor. If it is able to find the partner process, the corresponding link delays are added and the current process is put at the ready queue end (the text is available with the author).
There are several other procedures to add computational delay, communication delay, link delay, etc and procedure link send it changes the state of a process and removes it from the queue.
Simulate ( )
This module does the simulation work and a file is opened to write the instructions as executed by simulator. It starts on processor zero. If the time_stamp of current processor is greater than allotted time slice or if it is ready process queue then the procedure processor_schedule is allowed else procedure process_schedule is called.
The processor_scheduler finds the processor with minimum clock as the new current processor. It follows a linear search for the above purpose. The process_scheduler finds the process on the current processor having urgent priority and places it in ready_queue head of the current processor so as to execute it next (details available with the authors). The simulator continues till all instructions of all the processes are over.
Statistics ( )
This procedure calculates all the statistical information and stores them in a file. Time is estimated for the application to run on a single processor, the overall efficiency. Maximum of all processor clocks (details available with the author) is also calculated.

Algorithm

```
Begin
initialize(     ) // Initializes various parameters and variables.
read_input      // Read input from the designated file.
simulate        // Start the simulation.
statistics      // Transfer the desired results to a
                   predetermined file.
End
```

6. DISCUSSION

In this model, the processor executes a computation step and after finishing, they synchronize and perform data exchange in a cycle. If during execution of an algorithm, all the processors are performing computations in all cycles then the system utilisation is 1. However, it is found that in some algorithm all the processors may not participate in computation in all the cycles, as some processor may be waiting for the results generated by some other processors. The value for such algorithms is less than one.
The level of details required in the validation of a simulator should depend on how that simulator is to be used in decision making. If the performance measure thus obtained has some mean value (eg, CPU utilization, the average response time), then the notion of significance level and confidence interval should be applied to quantify the statistical significance of the difference between measured and simulated effects. The analysis of variance technique can also be used to test the hypothesis.

7. CONCLUSION

The model discussed here determines the performance of a static system. With some modifications, it can be made to work in dynamic environment also. The model discussed has got some limitations. Its advantage is that it helps smaller processes to complete execution by providing them time slices. In many cases the intermediate results provided by such processes is used by the other processes to continue execution. Since in most cases, parallel computers are used for similar kind of jobs repeatedly, by monitoring the communication pattern, the execution cycle can be varied to reduce the context switching overhead.

REFERENCES

[1]. SJ Akl. 'The Design and Analysis of Parallel Algorithms.' Prentice Hall, Inc, New Jersy, 1989.
[2]. H Alt, T Hagerup , K Mehlhorn and F P Preparata. ' Deterministic Simulation of Idealized Parallel Computers on More Realistic Ones.' SIAM Journal on Computing, vol 16,no 5, October 1987, p808.
[3]. A Basu, S Srinivas, K S Kumar, A Paulraj and LM Pattnaik.' Performance Analysis of algorithms on a

message Passing through Process.' 5th International IEEE Symposium on Parallel Processing,1991.

[4]. D Ferari. 'Computer System Performance Evaluation'. Prentice Hall, 1978.

[5]. JE Hall, DK Hsiao and HN Kamel. 'Performance Evaluation of a Parallel Sealable and Expandable Database Computer.' Proceedings of the Twenty fourth Annual Hawaii International IEEE Conference on System Sciences,1991.

[6]. Harvendra Kumar, A. K. Verma, "Comparative Study of Distributed Computing Paradigms", BVICAM's International Journal of Information Technology, BIJIT, issue 2, July-December 2009, Vol.1 No.2,

[7]. K Hwang and FA Briggs. 'Advanced computer Architecture and Parallel Processing.' McGraw Hill, New York,1989.

[8]. LH Jamieson, DB Gannon, RJ Douglas, (eds). 'The Characteristics of Parallel Algorithms.' The MIT Press 1989.

[9]. MJ Quinn. 'Designing Efficient Algorithms for Parallel Computers.' McGraw Hill, New York, 1987.

[10]. H A Sholl , y Quin and RA Ammar . 'Task Coalescence and Performance Evaluation for Parallel Software Structures on Distributed, Real-time System.' Processings of the Twentyfourth Annual Hawaii International IEEE Conference on System Sciences, 1991.

[11]. C R Snow.' Concurrent Programming.' Cambridge University Press,1988.

[12]. B W Stuck and E Arthurs. 'A Computer and communication Network Performance Analysis Primer. 'Prentice Hall, 1985.

[13]. L Svobodova. "Computer Performance Measurement and Evaluation Method: Analysis and Application", Elsevier, New York.

[14]. CD Howe and B Moxon. 'How to Program Parallel Processors.' Spectrum, vol 24, no 9, September 1987, p36

[15]. K. Bhatia, A. K. Pal, Anu Chaudhary, "Performance Analysis of High Speed Data Networks Using Priority Discipline", BVICAM's International Journal of Information Technology, BIJIT, issue 2, July-December 2009, Vol.1 No.2,