On Simulation of Massively Parallel Computers

Eduardo Rocha Rodrigues
Philippe O. A. Navaux
Federal University of Rio Grande do Sul - UFRGS
{errodrigues, navaux}@inf.ufrgs.br

Jairo Panetta
Brazilian Institute for Space Research - INPE
jairo.panetta@cptec.inpe.br

Abstract

High performance computers are employing more and more processors. Many centers worldwide have computers with hundreds or thousands of processors. Measuring performance of applications on these machines is highly desirable because one can use this information to adjust such applications to achieve better performance. However, the access to large machines is still restricted to few users. In this sense, simulators are a convenient way to measure and even debug programs intended for computers with many processors. In this text, we exam the state-of-art of simulators of large machines and present an experiment performed with the BigSim simulator.

1. Introduction

Currently, there is a trend towards increasing the number of processing units (cores) per chip. Companies like Intel, AMD and IBM have developed chips with up to 8 cores per die and it is expected this number will grow much more in the future. The high performance computers are also employing increasingly more processors, as it can be seen from the TOP500 list [1]. For example, the BlueGene/L in Lawrence Livermore National Laboratory (LNLL) has 131.072 processors and Earth Simulator has 640 nodes with 8 vector processors each. This fact imposes many challenges for development, debugging and management of programs executed on those machines. Undoubtedly, it will be needed to developed new methods, tools and strategies to deal with problems which may arise in those computers.

Although many centers worldwide already have large computers, it is quite difficult for most researcher to have access to these machines. Therefore, program development, testing and optimization for large computers are delayed. In addition, developing new technologies, such as new interconnection schemes, using real hardware can be really expensive. In this sense, simulators are a convenient way to measure and even test and debug programs intended for computers with large processor counts.

In this text, we are going to present some simulators which enable researchers to develop and test applications for large machines. Initially we will describe the type of simulation that is typically employed: Parallel Discrete Event Simulations (PDES). Afterwards, we will compare two of simulators, which are BigSim and *À la carte*. Finally, we will show an experiment using BigSim, since this simulator is available for test.

2. Parallel discrete event simulation

As stated before, simulations can be an alternative to test large computers when they are not available or before they are built. However, large machines probably cannot be simulated using a sequential approach. The simulation itself should be executed in parallel. Parallel simulations is indeed difficult and requires a way to synchronize the processors in charge of the simulation, so that causality errors do not occur. In this section, we describe a technique call Parallel Discrete Event Simulation (PDES) and some examples which employ this technique.

Discrete Event Simulation (DES) can be used to simulate systems whose behavior is represented by a chronological sequence of events. The sequential approach for this kind of simulations usually employs at least three different data structures [4]. (1) the state variables that represent the state of the real system, (2) an event list holding the events not yet processed, and (3) a global clock variable to denote the elapsed simulation time. Each event in the event list has a time stamp that denotes a change in the state variables. The time stamp correspond to the time when the real system state must be modified. Occasionally, an event can trigger new events for execution in the future. The simulation proceeds taking the smallest time stamp in the event list, changing the state variables and triggering new events accordingly.

The sequential approach for DES sometimes cannot deal with large simulations, due to time or resources restric-

tions. Large machine simulations with thousands of processors may fall in this situation. One possible strategy to solve this problem is to run the simulation in parallel, that is know as Parallel Discrete Event Simulation (PDES). However, trying to process events concurrently in different processors could cause causality errors, i.e. errors in the relationship between cause and effect. For example, consider two events E_1 and E_2 with timestamps T_1 and T_2 respectively and $T_1 < T_2$. If the event E_1 reads state variables that E_2 writes into, then this two events cannot be executed in parallel. More complex relations can arise and a mechanism is required to synchronize the processors.

Schemes to avoid causality errors can be classified basically in two categories: conservative scheme and optimistic scheme. The conservative scheme avoids the possibility of any causality error ever to occur. These approaches rely on some strategy to determine when it is safe to process an event. In its turn, the optimistic approach uses a detection and recovery mechanism to avoid causality errors, i.e. causality errors are detected and a rollback mechanism is invoked to recover. Many different mechanisms for both categories have been proposed, as can be seen in the article [4].

An example of simulator that uses the conservative approach is MPI-SIM [6]. This simulator can be employed to predict the performance of MPI programs using different architectural characteristics, as the number of processors and communication latencies. Two synchronization protocols (null message and conditional events) were combined in this simulator in order to reduce the frequency and cost of the synchronization among the execution of the events. However, it was used at most 16 processors to execute the simulation shown in the article. Another example of simulator that implements the conservative scheme is LAPSE (Large Application Parallel Simulation Environment) [3]. This simulator uses a distributed memory computer to simulate a message-passing code running on a large computer. But only experiments with 512 simulated processors are presented. A third example that falls in the conservative approach is Wisconsin Wind Tunnel (WWT) [7]. However it was developed specifically for evaluating shared memory computers.

An example of optimistic parallel discrete event simulator is Time Warp Operating System (TWOS) [5]. It was designed to execute on multiple nodes of a parallel processor, or on several Sun workstations connected by an Ethernet. TWOS uses an optimistic synchronization method that incorporates a full rollback mechanism. TWOS is a generic PDES simulator, that can be used to simulate many different problems.

3. A comparison between two large scale machine simulators

This section presents two simulators which employ PDES to specifically simulate large computers. The first simulator is *À la carte* simulation framework, developed in Los Alamos National Laboratory, and the second is BigSim/BigNetSim, developed at the University of Illinois at Urbana Champaign.

3.1. *À la carte*

À la carte is an approach for simulating computing architectures applicable to extreme-scale systems (thousands of processors) and to advanced novel architectural configurations [2]. It was built using a component-based design that allows a seamless assembly of architectures from representations of workload, processors and network devices. It was designed to support studies on software scalability and properties of the systems themselves.

The objectives of this simulator are: (1) study of different hardware / architecture design; (2) study of algorithms and its corresponding implementation at the application and system levels; (3) determine the system scalability with the number of processors and other components like network communication devices; (4) analysis of the trade-offs between performance and cost; and, (5) testing and validating analytical models of computation and communication.

The simulator is divided in three parts: component descriptions; configuration descriptions; and an underlying simulation system, that is a PDES system. Components in this simulators may be processors, switches, network interfaces and applications workloads. *À la carte* uses the Dartmouth Scalable Simulation Framework (DaSSF). DaSSF was developed as a common parallel simulation API and employs a conservative scheme to deal with causalities errors. DaSSF manages the synchronization, scheduling, and delivery of events in the simulation. It has a C++ API and supports both shared-memory and distributed-memory parallelism.

The objective of the *À la carte* project is to simulate computing platforms like ASCI systems of the Department of Energy of the USA. However, there are not performance results of simulations with more than 250 simulated processors. Furthermore, the simulator is not available for testing.

3.2. BigSim/BigNetSim

BigSim/BigNetSim is a performance prediction environment for large scale computers [8]. BigSim is the parallel simulator for predicting performance of machines with a very large number of processors and BigNetSim incorporates a pluggable module of a detailed contention-based

network model. This simulator uses Charm++ and AMPI as the programming model for large scale computers. This languages are very convenient since it hides from the user the decision of allocation of tasks to processors.

Charm++ is an oriented object and parallel language based on C++ and an execution environment. A program written in Charm++ is made of a collection of distributed objects whose methods can be called asynchronously. The execution environment controls the object distribution and redistribution among the processors and intermediates their communications. This scheme allows many optimizations such as load balancing, overlap between processing and communications and checkpoint/restart mechanisms. AMPI is a Message Passing Interface (MPI) implementation built on top of Charm++. In this way this implementation allows MPI programs adapt its execution to achieve load balancing.

In order to simulate large machines BigSim emulates a computing node as a set of threads with a common shared memory. This threads are classified as communication threads and working threads. A runtime library built on top of this abstraction can send a message to a destination node. The message contains a handler function to be invoked at the destination. Communication threads in a node verify incoming messages and put them in a global buffer or a local buffer of a worker thread. The worker threads repeatedly retrieve messages and execute the corresponding function. As stated in the article [9], this model is general enough to encompass a wide range of different possible architectures. On top of this abstraction it was implemented a AMPI and Charm++ version, so that the user can programs as if it is using a large machine with many of this nodes.

It is clearly impractical, if not impossible, to simulate thousands of processors in a single processor. But, as parallel applications can be described as actions occurring at a particular time and lasting for a known duration, PDES can be used to simulate this behavior. However, this approach has to deal with the complexity of the communication, since message in this environment may arrive out of order, arising from the fact that the simulation are using multiple processors. In order to deal with these causality errors BigSim lets the simulated applications proceed as usual, while concurrently running a parallel algorithm that corrects time-stamps of individual messages. Since the applications are usually deterministic, this solutions avoids the high costs of rollbacks.

Different degrees of accuracy can be used in BigSim to simulate the sequential blocks of the application code. The user can supply an expressions to each block of code estimating the time of its execution. This alternative has the advantage of flexibility, but it burdens the user with the task of accurate estimation. Another possibility is using the wallclock multiplied by a factor to obtain the estimated execution time of the block on the target machine. A third alternative is use hardware counter and a heuristic to estimate the performance on the target machine.

BigSim can also generate logs of sequential computation blocks, the messages and the dependencies among blocks. Thus it is possible simulate more complex network topologies and contention models using the BigNetSim. The logs are read by BigNetSim which simulates the execution of the original tasks by elapsing time, satisfying dependencies, and spawning additional tasks by passing messages through a detailed network contention model.

Experimental results are presented using up to 32.000 simulated processors to execute a finite element method software. A molecular dynamic simulation was also executed on the simulator. In order to validate the approach, some results were compared with a real BlueGene/L computer [8].

The Table 1 summarizes a comparison between some characteristics of both BigSim/BigNetSim and *À la carte* systems.

4. The experiment

In this section, we will present an experiment in which we have used BigSim to simulate four hundred processors using only forty real processors. The algorithm employed in this test was the one to find the shortest path in a graph. We then compare the results with the theoretical performance model.

A well known algorithm to find the shortest path in a graph is the Floyd's algorithm. Its basic operation is to determine a path going from a vertex v_i to v_j passing through v_k such that the cost is lower than going directly from v_i to v_j. This type of algorithm has many application in communication, transport and electronic problems.

The parallel version employs a domain decomposition approach. Given an adjacency matrix with dimension N, which represents a graph, each one of the P processors takes a block (with dimension $N/\sqrt{P}$) in which it computes the basic operation. The vertex v_k may not reside in a certain processor, therefore, a communication is required.

The parallel performance can be modeled by means of a simple latency approach. This model considers that each message requires a certain time to arrive its destination. In this way, in order to model the Floyd's performance it is necessary to count the number of message sent during its execution. In addition, it is also necessary to count the number of basic operations which is executed.

The time spent with the basic operations is proportional to N^3/P because it is considered N vertices v_k for each pair in the process's block. The communication requires two broadcasts for each vertex v_k, one broadcast among the processors in the same column and another among the proces-

Simulator	Parallel	Synchronization	accuracy	Application
BigSim/ BigNetSim	PDES	Optimistic	user-supplied direct execution	MPI Charm++
À la carte	PDES	Conservative	direct execution	MPI

Table 1. Comparison between BigSim/BigNetSim and *À la carte*.

sors in the same line. Therefore the parallel performance is given by:

$$t_c N^3/P + 2Nlog\sqrt{P}t_s$$

where t_c is the basic operation cost and t_s is the network latency.

We used the BigSim to execute the program which implement the Floyd's algorithm. We developed this program with the MPI standard so that we could execute it with the AMPI library. The benefit of this approach is that we could compile and run the same code which can be executed in a real machine.

In order to compare both the theoretical model and the simulation, we choose different adjacency matrix sizes as input. We feed the model with the simulated latency and the cost for each basic operation. The final result can be seen on Figure 1.

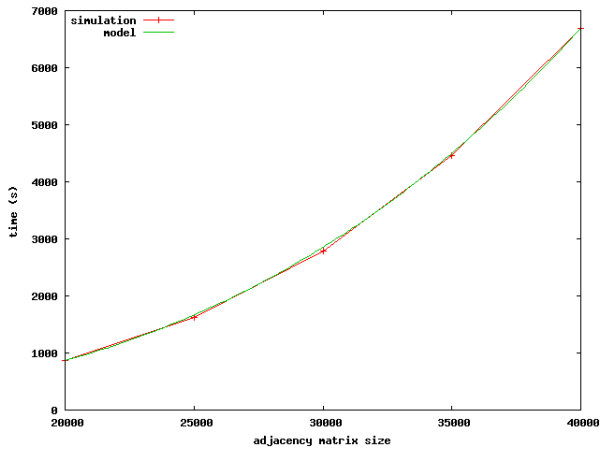

Figure 1. Comparison between the theoretical and simulated time.

This result shows that the BigSim can be used to predict the performance of a program as if it were executed in a large machine. This environment, therefore, is able to support the development of new algorithms and models which are been developed in the authors' research project.

5. Final remarks

In this text, we presented a technique and some simulators to simulate the behavior of large computers. Since it is expected that processor counts will increase in the near future, such environments are useful for research of new algorithms and models.

Here, we also presented a result of simulation in which we used the BigSim to execute a shortest-path algorithm. The objective was to verify if this simulator could be used as an environment for future experiments. In this sense, this software seems to be suitable for the our purposes, which is to develop load balancing algorithms for large machines.

The next steps include to simulate other applications and adapt the BigSim to simulate the common network interconnect Ethernet.

References

[1] Top 500 supercomputing sites, http://www.top500.org/, 2007.

[2] K. Berkbigler, B. Bush, K. Davis, N. Moss, S. Smith, T. Caudell, K. Summers, et al. Á la carte: A Simulation Framework for Extreme-scale Hardware Architectures.

[3] P. M. Dickens, P. Heidelberger, and D. M. Nicol. A distributed memory LAPSE: Parallel simulation of message-passing programs. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, 1994.

[4] R. M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, 1990.

[5] D. Jefferson, B. Beckman, F. Wieland, L. Blume, and M. Diloreto. Time warp operating system. In *SOSP '87: Proceedings of the eleventh ACM Symposium on Operating systems principles*, New York, NY, USA, 1987. ACM.

[6] S. Prakash and R. Bagrodia. MPI-SIM: Using parallel simulation to evaluate MPI programs. In *Winter Simulation Conference*, pages 467–474, 1998.

[7] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood. The wisconsin wind tunnel: Virtual prototyping of parallel computers. In *Measurement and Modeling of Computer Systems*, pages 48–60, 1993.

[8] G. Zheng. *Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing*. PhD thesis, Dept. of Computer Science, UIUC, 2005.

[9] G. Zheng, T. Wilmarth, P. Jagadishprasad, and L. V. Kalé. Simulation-based performance prediction for large parallel machines. In *International Journal of Parallel Programming*, volume 33, pages 183–207, 2005.