2010 Ninth International Conference on Grid and Cloud Computing

A Parallel Simulator for Large-Scale Parallel Computers

Yuzhe Zhi, Yi Liu
Sino-German Software Institution
Beihang University
Beijing, P.R.China
{yuzhe.zhi, yi.liu}@jsi.buaa.edu.cn

Lin Jiao, Peng Zhang
Department of Computer Science
Xi'an Jiaotong University
Xi'an, P.R.China

***Abstract*—This paper describes the design and application of an execution-driven parallel simulator for predicting performance of Large-Scale Parallel Computers. The simulator can be used in hardware validation and software development for large-scale parallel computers. It simulates processors of each node, network components and disk I/O components. To illustrate the capabilities of our simulator, we describe experiments running the High Performance Linpack Benchmark on a simulated parallel computer. It gains sound speedup and executes fast enough to run complex MPI codes.**

***Keywords*—simulator, parallel simulation, execution-driven, large-scale parallel computer**

I. Introduction

Simulation is an important tool for performance evaluation of computer systems. Simulators reduce the cost and time of a project by allowing the architect to quickly evaluate the performance of a wide range of designs. In order to meet the growing need of variety applications, large-scale parallel computers with extremely large number of processors are now being designed. During development of a large-scale parallel computer, there are many significant factors should be considered and architecture simulation is an important research approach. However, the trend in parallelism of computer system imposes great challenges to traditional sequential simulators [1,2,3]. Low execution speed of a sequential simulator cannot satisfy the requirement of a large-scale target system since it exploits the computing capacity of no more than one processor.

Parallel simulation can largely reduce the simulation time and take good use of existing parallel computing platforms. But existing simulators for computer systems were either small-scale [1] or large-scale for specific architecture [4]. So far as we know, few large-scale simulators that simulated all components including processors, network and disk I/O have been published.

The main contributions of this work are outlined as follows:

- We design a parallel simulator for all instruction set architecture as long as the host and target machine nodes have the same architecture.
- All the components of the target system are simulated, including processors of each node, network interconnection system and disk I/O system.
- We propose a novel scheme for task allocation, and implement a synchronize algorithm based on Lamport [5,6].

This paper is organized as follows. The next section presents the related work in parallel simulation framework. Section III introduces the architecture of our simulator and presents our simulator in detail. Section IV describes experiments running the High Performance Linpack Benchmark (HPL) [7] on a simulated parallel computer and Section V gives the conclusions.

II. Related Works

SimpleScalar [8] is a kind of computer system simulation and modeling tool which has been used widely in the computer architecture research field. It can emulate the Alpha, PISA, ARM, and x86 instruction sets. Because we want to use a parallel simulator to evaluate the performance of large-scale computers by simulate all the component of the computers, SimpleScalar is not appropriate. Our simulator models all architected features of large-scale parallel computer, including network interconnection and disk I/O component.

BGLsim [9] is a simulator developed by IBM for BlueGene/L, which runs on a Linux cluster. It executes fast enough to run complete operating system and complex MPI codes. BGLsim accurately models all architected feature of the BlueGene/L hardware at the instruction-set level. However, BGLsim mainly focus on PowerPC architecture and its instruction set level simulation costs time a lot. In contrast, our simulator can simulate any architecture as long as the host and target machine nodes' architecture are same. Besides, in order to get high efficiency, our simulator does not simulate in instruction set level.

MPI-SIM [10] is a library for execution-driven parallel simulation of MPI programs. It aims at evaluating the performance of exiting MPI programs in large-scale systems, uses a novel conservative synchronization algorithm. But MPI-SIM does not provide access to the underlying layers of communication and interconnection devices. It is not a full simulator that can model a target system precisely. Since we

This work is supported by National Hi-tech Research and Development Program of China under grant No.2009AA01Z107, 2009AA01A129 and National Science Foundation of China under grant No.60873053...

want to find a simulator that can simulate all the module of the target system, MPI-SIM is not the complete solution we are looking for.

BigSim [4] is a parallel simulator for predicting performance of machines with a very large number of processors. It is based on the CHARM++ [11] parallel programming system, and can emulate several million threads on clusters with only hundreds of processors. But BigSim relys on CHARM++ environment too closely limits it be widely used.

We decide to develop our own simulator for large-scale parallel machines. This paper propose a novel scheme for task allocation, and implement a synchronize algorithm based on Lamport [5,6].

III. Simulator Overview

Speed is the primary constraint for simulator usage. The computer system is becoming more and more complex, however improves of the simulator speed slowly. In our simulator, we do not simulate the processors in instruction set level. Since lots of the large scale computers are assembled by using the existing processors, our simulator assumes that the processors of host machine and target machine are the same. This approach can largely enhance the efficiency, because the execute time of certain task is equal in both host and target systems. For this reason, we can pay more attention to the network and disk I/O component simulation. Fig. 1 shows the simulator architecture.

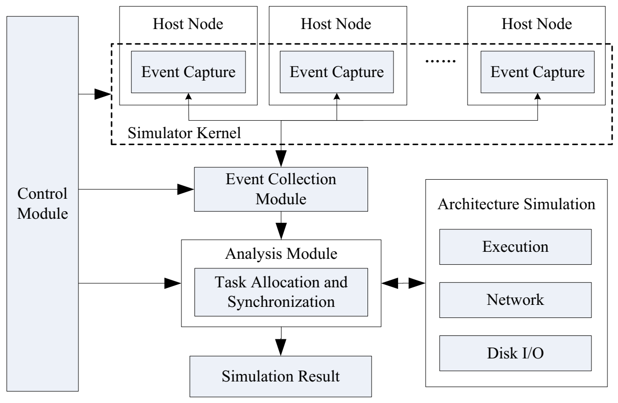

Figure 1. Simulator architecture

As shown in Fig. 1, the simulator is a parallel simulator by using cluster as host machine. Besides, the simulator uses execution-driven approach, means the MPI applications execute directly on the simulator. The event capture module catches the execution, network and disk I/O event we interest while applications run, and event collection module collect all the event captured from different host nodes. Analysis module gives the simulation result by allocate and synchronize the task properly. Architecture simulation module gives the time of each event in target machine.

At the end of each simulation, our simulator provides a performance statistic form which can helps architect to analyze the performance characteristics of applications and target systems. The fallowing subsection will introduce the key technology of the simulator in detail.

A. Event Capture

Event capture module catches the execution, network and disk I/O event we interest while applications run. Performance of the target system can be predicted by analyzing these captured events. This subsection introduces how we capture these events.

In Linux system, lifetime of a process can be divided into three parts. As shown in (1).

$$T_{process} = T_{run} + T_{IO} + T_{ready}. \tag{1}$$

$T_{process}$ is the time span of a process from created to terminated, T_{run} is the execution time on processor, T_{IO} is the block time caused by I/O operations, T_{ready} is the waiting time in ready queue.

T_{run} is the time of a process running on the processor. As the process is repeatedly switched onto and out of the processor, thus, the T_{run} consist of many time slices. Time slice starting with the process switched onto processor, and ending with the process blocked by a higher priority process or I/O operations. So, T_{run} can be calculated as long as we record the moment these events occur. According to the implementation mechanism of Linux kernel, when these events occur, Linux kernel function schedule () will be called. So, these events' occur time can be traced by tracking the schedule () function.

T_{IO} can be divided into network communication time and disk access time. When an I/O event occurs, simulator calls network or disk I/O simulation module with proper parameters. Parameters of network module are source and destination node number, message length, etc. Parameters of Disk I/O module are file name, read-write mode, data length, etc. These parameters can obtained by capture the corresponding Linux system calls such as send(), recv(), sys_read(), sys_write(), etc.

For T_{ready}, it can be calculated according to the I/O operation time T_{IO}, switching time of the process switched onto and out of the processor recorded during the calculation of T_{run}, etc.

B. Task Allocation

Parallel simulators always run several million processes or threads, so how to efficient allocates and maps all these tasks from host computer system to target computer system significantly affect the performance of simulator. In the simulation of the target system, we cannot map the processes to certain target nodes randomly. We need a process allocation algorithm to reasonable map the application processes of the host system to target system.

The method of mapping processes from host system to target system should meet the prerequisites as follow:

- Ultimately, the processes of certain target node belong to the same host node.
- Workload of each target node is balanced.

Based on the prerequisites above, the task allocation algorithm including two steps:

- Complete the mapping from host node to target node.
- Complete the mapping from host node processes to target node processes.

After step 1, a host node will be mapped to many target nodes. After step 2, many processes of one host node will be mapped to the target nodes which this host node simulate, thus meets the prerequisite 1. Besides, in order to ensure that the loads of all target nodes is balanced, the situation of one host node simulate many target nodes and other host nodes simulate only one should be avoid.

The task allocation algorithm is represented in pseudo-code in Fig. 2. N_{host} represents the host number of the simulator, N_{node} represents the node number of the target computer been simulated, and N_p is the process number in each target node.

```
/*step 1, allocates $N_{node}$ target nodes to $N_{host}$ host nodes*/
1.  **Operation** ($N_{node}$, $N_{host}$);
/*step 2, allocates processes in each host node to target
nodes*/
1.  $N_{nph} = N_{node} / N_{host}$;  //target node number in each host
    node
2.  $N_{pph} = N_p * N_{nph}$; //process number in each host node
3.  **for** (each host node){
4.      **Operation** ($N_{pph}$, $N_{nph}$);
5.  }
/*two steps above call the same procedure as follow*/
1.  **procedure Operation**(tasks, components){
2.      tasks of each component = tasks / components;
3.      i = 0;
4.      /*allocate task for component i*/
5.      **while** (i < components){
6.          j = 0;
7.          **while** (j < tasks of each component){
8.              **if** (have task unallocated){
9.                  allocate present task to i component;
10.                 j++;
11.             }**else**
12.                 allocate complete;
13.         }
14.         i++; //allocate next component
15.     }
16. }
```

Figure 2. Task allocation algorithm

C. Synchronization

Execution of the parallel simulator should be properly synchronized. This is particularly important in the veracity of the simulation. In distribution system, synchronization not only ensures that events are processed in a correct order, but also helps to ensure that the events execute in reasonable logical sequence. For example, when one process is waiting for a massage, it should not execute until received the certain message.

Synchronization mechanism, can be classified as being either conservative or optimistic, usually assume that the simulation consists of logical processes (LPs) that communicate by exchanging time stamped massages. Optimistic synchronization needs log the execute process. When an LP receives an event with time stamp smaller than one or more events it has already processed, it rolls back and reprocesses those events in timestamp order. In parallel simulation, log and rollback will consume unacceptable huge CPU and memory resources. So, conservative mechanism is preferred in this paper.

The synchronize mechanism in this paper is based on Lamport. The principal task of synchronization is to determine when it is safe to process an event. According to the first come first serve scheduling of Lamport, each LP execute the event with the smallest timestamp. This can ensure that no event with timestamp less than the current event timestamp been processed. An LP cannot process an event when it is waiting for a message. In order to synchronize the communication between LPs, each LP holds a queue to record the receive events. Receive event means that the LP is waiting for a message sent by other PLs. Events in the queue are ordered by timestamp. If the timestamp of current event is small than the receive event at the first of the queue, the event can be processed; otherwise, the LP blocked until receive the corresponding message from other PLs. This can ensure the sequence of events executed in reasonable order. The delay of message send from one PL to another should be simulated by network model. The following two sub-sections will introduce the network and disk I/O simulation.

D. Network Simulation

Simulator calls the network simulation module to get the network communication delay. The target machine is multi-core processor based large-scale parallel computer. Nodes and disk arrays of the computer realize the communication with each other through message passing, thus the number of network communication message is very large. If we detailed simulate every network event, even if the application to simulate very small, it takes a long simulation time. This led to a serious decline to simulation system efficiency. Therefore, we adopt modeling method to realize a mathematical model of interconnection networks, which can calculate the corresponding network delay according to the parameters of network events such as source and destination node number, message length, etc. This method not only improves the simulation efficiency, but also ensures a certain degree of simulation accuracy. It's an appropriate balance of the efficiency and accuracy.

The target computer system of this paper will use InfiniBand [12] interconnection network. After abstraction and analysis of the InfiniBand network, we established InfiniBand switch model, node model, the path model and packet network delay model respectively, and finally got a network delay mathematical formula. Then take advantage of the network delay mathematical formula, we analyzed 2D-mesh and fat-tree topology InfiniBand networks respectively. The performance of the two topology networks will illustrate in Section IV.

E. *Disk I/O Simulation*

Storage device of the target computer system mainly compose of disk arrays which runs parallel file system on it. The computing nodes and disk arrays are connected through the interconnection network. Therefore, time consume of read and write requests can be divided into two parts: (a) interconnection network delay that the read and write requests reach the disk array; (b) time required to access the disk array. The network delay can obtain from the interconnection network simulation module and disk array access time can obtain from the disk simulation module introduced below.

Main task of disk simulation is the models of the disk array access delay. The disk array access delay including the file system access time, waiting time in the case of read-write dependence, disk block allocation time, bus request time, seek time, rotational latency and data transmission time. The disk simulation realized by model all the time above respectively.

IV. Case Studies

We first present the result of validation of the simulator on our existing computer. Then we present results of performance prediction of a large-scale machine using the nodes consists of Godson CPU. Finally we present the slowdown of our simulator to illustrate its high simulation efficiency.

A. *Validation*

HPL, a portable implementation of the High-Performance Linpack Benchmark [7], is used as a performance measure for ranking supercomputers in the TOP500 [13] list of the world's fastest computers. HPL is self-checking, it reports whether the calculation completed successfully by looking at the residues.

In order to validate our simulator, we chose to run HPL on one host node to simulate 1, 2, 4, and 6 target nodes, and compared the simulation result with the actual HPL performance. The result is shown in Table 1 for a problem with sixed size in all the runs. The first row shows the HPL performance on 1 to 6 nodes; the second row shows the predicted HPL performance when simulating these nodes using only one host node. It shows the predicted and measured speed differed by less than 10%, and lending reasonable credibility to the simulations.

TABLE I. Actual vs. predicted speed

Nodes	1	2	4	6
Actual speed (Gflops)	1.516	2.844	4.442	5.026
Predicted speed (Gflops)	1.509	2.83719	4.86714	5.44233

B. *Performance Prediction*

We are now able to study the performance of a machine before it built. Our target computer system will use Godson 3B CPU and Infiniband network. Godson, also known as Dragon chip, is a family of general-purpose MIPS-compatible CPUs developed at the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS) in China. Because of the developing of Godson 3B still not finished, we cannot get host nodes same as the target nodes. We simulating the execute time of the nodes via a suitable multiplier (scale-factor) [4]. The scale-factor is mainly the ratio of frequency of different CPUs.

The simulation system in this paper consists of 4 nodes IBM BladerCenter. In order to evaluate the performance of large-scale computer, we simulated 32, 64, 128, 256, 512 and 1024 nodes target machines respectively. We also evaluated the performance of fat-tree and 2D-mesh network. Fig. 3 shows the HPL performance of the target machines in different network configuration. Fig. 4 shows the average communication time per node of each network topology.

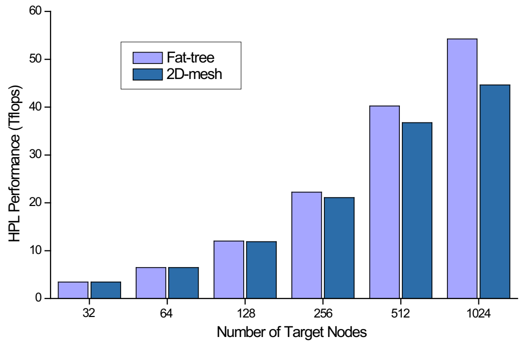

Figure 3. HPL performance of target computers of Fat-tree and 2D-mesh topology

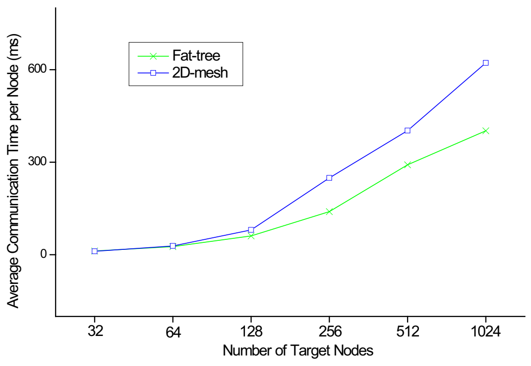

Figure 4. Average communication time per node of Fat-tree and 2D-mesh topology

As can be seen from Fig. 3, the fat-tree interconnection network topology is superior to 2D-mesh topology in the same scale. Fig. 4 further illustrates the causes of the difference of HPL performance in different interconnection network configuration. With the node size increases, 2d-mesh topology network communication needs much greater delay than the fat-tree topology. The likely reason is fat-tree topology with high efficiency in local communication, because the links in a fat-tree become "fatter" as one moves up the tree towards the root. In contrast, the 2D-mesh topology routing algorithm is relatively simple, and cannot be tailored to specific packaging technologies.

C. Slowdown

In order to analyze the slowdown introduced by our simulator in the execution of an application, we compare the execution time of the benchmark when running natively on our host machines and when running on the simulated target machine. Fig. 5 presents the slowdown, ratio of simulation time and native time. The slowdown is just 127 when simulate a 1024 nodes target machine, this indicate that our simulator have higher efficiency than many exiting simulators. For example, the slowdown of parallel simulator BGLsim is 220 at least [6] when simulating with 4 host nodes.

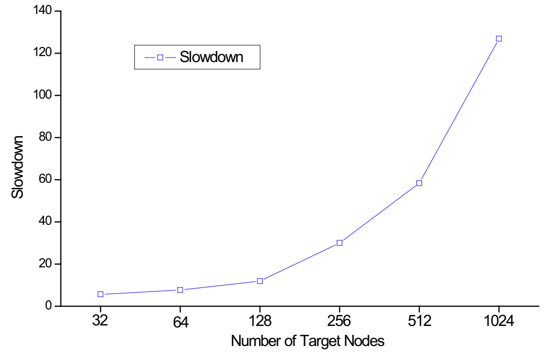

Figure 5. Slowdown of the simulated HPL Benchmark

V. CONCLUSIONS AND FUTURE WORK

The parallel simulator introduced in this paper adopts distributed simulation infrastructure, which make the simulator scalable and efficient. The applications based on MPI interfaces can be executed directly on the simulator without any modifications. The simulator can be used for software development and hardware analysis. Experiments were performed to validate the functionality of the simulator and measure simulation speed.

The simulator is not cycle accurate and assumes the target and host nodes with same instruction architecture. For the different instruction set, we simulated via a suitable multiplier. Although not simulate the instruction-set, we believe the simulation models can have enough accuracy to support design decisions, in both hardware and software. It is useful for verifying the architectural design of a large-scale parallel computer before it is deployed. Future work will focus on increasing accuracy by incorporating an instruction level simulator.

[1] T Austin, E Larson, D Ernst. Simplescalar: An infrastructure for computer system modeling [J]. IEEE Computer, 2002, 35(2):59-67 ;

[2] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg amd Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A full system simulation platform[J]. IEEE Computer, 35(2):50–58, February 2002.

[3] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, Complete Computer Simulation: The SimOS Approach[J]. Parallel and Distributed Technology, vol. 3, no. 4, pp. 35-43, Winter 1995.

[4] Zheng G，Kakulapati G，Kale L．Bigsim : A parallel simulator for performance prediction of extremely large parallel ma,-chines． In Proc ． 18th International Parallel and Distributed Processing Symposium，Santa Fe，USA，April 26-30，2004，P．786．

[5] L Lamport. Time, clocks, and the ordering of events in a distributed system[C]. Communications of the ACM, 1978 21(7):558-565.

[6] L Lamport. Concurrent reading and writing of clocks[C]. ACM Transactions on Computer Systems (TOCS) , 1990, 8(4):305-310.

[7] HPL Home Page. http://www.netlib.org/benchmark/hpl/

[8] http://www.cs.wisc.edu/~mscalar/simplescalar.html

[9] Ceze L, Strauss K, Almasi G, Bohrer P, Brunheroto J, Cascaval C, Castanos J, Lieber D, Martorell X, Moreira J, Sanomiya A, Schenfeld E. Full circle: Simulating Linux clusters on Linux clusters. In Proc. the Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003, 2003.

[10] Prakash S, Bagrodia R L. MPI-SIM: Using parallel simulation to evaluate mpi programs． In Proc. the 30th Conference on Winter Simulation(WSC 1998), Los Alamitos, USA, Dec. 13-16, 1998, PP.467−474.

[11] L. V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In G. V. Wilson and P. Lu, editors, Parallel Programming using C++, pages 175–213. MIT Press, 1996.

[12] Infiniband Architecture Specificantion (1.21), Infiniband Trade Assoc., http://www.Infinibandta.com/, Nov. 2007.

[13] TOP500 Supercomputing Sites. http://www.top500.org/