SimNUMA: Simulating NUMA-Architecture Multiprocessor Systems Efficiently

Yi Liu, Yanchao Zhu, Xiang Li, Zehui Ni, Tao Liu
Sino-German Joint Software Institute
Beihang University
Beijing China
yi.liu@jsi.buaa.edu.cn, zyc0627cool@163.com

Yali Chen, Jin Wu
Huawei Technologies Co. Ltd
Shenzhen China

***Abstract*—Non-uniform memory access (NUMA) architecture is widely used in high-end servers and computing systems due to its scalability. In recent years, the number of processor cores in NUMA systems increases rapidly with the development of multi-core processors. Along with the growing of system scales, simulation of NUMA systems becomes a challenge to traditional general-purpose simulators by reason of their low simulation performance. This paper presents SimNUMA, an execution-driven full-system simulator dedicated for NUMA systems. In the design of SimNUMA, to improve simulation performance significantly, the same type of processor with the target machine is used in the host system, and a new method to capture remote-memory accesses efficiently is proposed, in addition, parallel simulation is used to achieve scalability and improve performance. The modeling and simulation of interconnection networks are also supported. The simulator is tested in accuracy, scalability and performance; results show that the simulation slowdown is rather satisfying. Finally, the paper gives simulation experiments for different scales of target NUMA systems.**

Keywords—simulator; NUMA architecture; remote-memory access; multi-core processor; execution-driven

I. INTRODUCTION

Non-uniform memory access (NUMA) architecture is more scalable than symmetric multi-processing (SMP) architecture, and widely used in high-end servers and computing systems. In the era of cloud computing, to meet requirements of some high-end cloud users and provide more elastic on-demand computing services, sometimes it is needed to integrate more computing power in a single machine instead of distributed memory systems. Under this background, it is necessary to exploit NUMA-architectures that can be scaled up/down flexibly in modular based on commercial interconnection technologies such as Intel QPI and Infiniband. On the other hand, with the development of multi-core processors, the number of processing cores increased rapidly, e.g. a 64-way server with quad-core processor has 256 cores, and a 128-way server with eight-core processor will have 1024 cores. Simulation of such kind of systems is very challenging to traditional general-purpose simulators such as Simics and Gems, due to their low simulation efficiency.

This paper presents SimNUMA, an execution-driven full-system simulator dedicated for NUMA systems. To improve simulation performance significantly, the simulator uses the same type of processor with the target machine in the host system, and proposes a new method to capture remote-memory accesses efficiently, in addition, parallel simulation is used to achieve scalability and improve performance. The modeling and simulation of interconnection networks are also supported. The system is tested in validation, scalability and performance; results show that the simulation slowdown is rather satisfying. Finally, the paper gives simulation experiments for different scales of target NUMA systems.

The rest of this paper is organized as follows. Section II gives an overview of related works and summarizes characteristics of our simulator. Section II analyzes critical simulation methods of the SimNUMA. Section IV introduces the architecture of the simulator. Section V tests and evaluates the system with benchmark applications. And Section VI concludes the paper.

II. RELATED WORKS

Simulation plays an important role in the research of computer architecture. In the past years, a bunch of simulators have been developed, such as classical SimpleScalar[1], SimOS[2], and recently emerged Simics[3], GEMS[4] and M5[5]. Most of them support execution-driven simulation, and some of which can provide full-system simulation. As general-purpose simulators, they generally focus on detailed modeling and simulation of processor's micro-architectures and memory hierarchy, such as the pipeline, cache hierarchies, etc. Although these powerful functions are very useful in exploitation of various kinds of new architectures, the detailed modeling and simulation reduces efficiency seriously. Typically the slowdown factor of these general-purpose simulators is more than 1,000 times, sometimes even more than 10,000 times, which depends on how detailed the simulation is.

Simulators dedicated for specific architectures have also been developed, e.g., tools for shared memory and NUMA architecture - RSIM[6],[7], SICOSYS[8], and SIMT[9]. RSIM (Rice Simulator for ILP Multiprocessors) is an execution-driven simulator for multiprocessor systems with directory-based cc-NUMA, which supports a number of shared memory multiprocessors. However, RSIM is not a full-system simulation tool, that is, it cannot support startup and execution of operating systems. Since it emphasizes on accuracy, RSIM

1521-9097/13 $31.00 © 2013 IEEE
DOI 10.1109/ICPADS.2013.55

is slower than simulators that exclude detailed processor modeling. Moreover, RSIM can only simulate mesh network and directory-based cache coherence mechanism. SICOSYS is a general-purpose interconnection network simulator, and can be regarded as an enhancement of RSIM on interconnection networks. The final simulator is an integration of SICOSYS with RSIM, and the simulation results show that it has similar accuracy with VHDL simulators and lower computational cost than RSIM. SIMT is a simulation tool for NUMA machines which relies on another tool Augmint to provide memory reference data, and mainly focuses on memory hierarchy such as cache misses and cache invalidations. While in a certain degree, it ignores interconnection networks since it just uses constants as communication latencies, in addition, it just uses estimated time to model the execution time of instructions.

Compared to current available simulators, the SimNUMA presented in this paper is an execution-driven, full-system simulator dedicated for NUMA-architecture systems. The main characteristics of the SimNUMA include: it uses the same type of processor with the target machine in the host system to avoid complex and inefficient modeling and simulation of micro-architecture and instruction-set for commercial available processors; on the basis of that, a new method to capture and simulate remote-memory accesses efficiently is proposed; in addition, parallel simulation is supported to achieve scalability. Consequently, the simulation performance is improved significantly. Moreover, modeling and simulation of different kinds of interconnection networks are supported in SimNUMA.

III. SIMULATION METHODS

A. Analysis

As mentioned in Section I, along with increasing of processing cores in NUMA systems, the simulation of such kind of systems turns to be challenges to traditional general-purpose simulators. The reason is that these simulators focus on the modeling and simulation of micro-architectures of processors, such as pipeline and cache, and in addition, to support simulation of different kinds of processors, instruction-set must be modeled and executed in simulation mode. As the result, the execution of applications in simulator is far slower than in target systems. Generally, the slowdown factor is more than 1,000 times, sometimes more than 10,000 times, depending on how detailed the simulation is.

To achieve significant improvement in simulation performance, SimNUMA uses the same type of processor with the target machine in the host system. The reason for this decision is that the vendors of NUMA systems generally use commercial processors in their machine instead of design new processors by themselves. Since the processors are the same between host and target, the execution time of the same instruction sequences will also be the same without considering memory accesses, so the modeling and simulation of processor micro-architectures and instruction-set can be avoided in the simulator. As the results, the simulation performance can be improved significantly. Another method to improve performance is parallel simulation, that is, use multiple multi-core processors (e.g. SMP system) to simulate target NUMA systems.

The primary distinct feature of NUMA system is that its memory are global addressed but distributed in each processing nodes, leading to different latencies for different memory locations. In NUMA systems, memory accesses can be divided into two categories: local-memory and remote-memory accesses, furthermore, latencies of remote-memory accesses are also different, depending on the length of transport route and congestion status of the interconnection network. Considering this, the remote-memory accesses must be identified and simulated in our simulator.

The most primary goal of a simulator is to obtain the execution time of an application in target machine, furthermore, execution time of processes and threads in this application. The simulation method of this execution time is analyzed as follows, and to simplify the discussion, the term ***process*** is used to denote the concurrent unit of operating systems which may be process or thread.

In target systems, application processes are scheduled by operation system, and at any time, they stay at either state of "*running*", "*waiting*" or "*blocked*" until they are terminated. So the execution time of a process can be calculated by:

$$\begin{aligned} T_{process} &= T_{run} + T_{IO} + T_{ready} \\ &= (T_{inst} + T_{rmem}) + T_{IO} + T_{ready} \end{aligned} \quad (1)$$

where T_{run} is the time the process stayed at "*running*" state, in other words, the time the process really executed on a processor; T_{IO} is the time the process stayed at "*blocked*" state, generally due to an I/O operation; and T_{ready} is the time the process stayed at "*waiting*" state, generally due to time-slice scheduling. If latencies of remote-memory accesses are considered, T_{run} can be further divided into two parts: the execution time of instructions including local-memory accesses (i.e. T_{inst}) and time of remote-memory accesses (i.e. T_{rmem}).

As mentioned above, the same type of processor is used in host and target, so the execution time of an instruction sequence is the same between host and target, that is, the T_{inst} of processes are the same. Unfortunately, this conclusion is not effect to T_{rmem}, T_{IO} and T_{ready}, since the number of processors, the interconnection network and storage system are different between host and target. These three parts of time can only be calculated on the basis of modeling and simulation of target system.

1) Based on the above analysis, to simulate execution time of application processes, and obtain statistics about remote-memory accesses, inter-process communications and I/O operations, the core tasks for our simulator include: Event capture: capture concerned events during the execution of processes, including remote-memory accesses, process scheduling, inter-process communication and file access operations. According to the captured events of process scheduling, the T_{inst} of each process can be counted, the other captured events can be used to drive the simulation of target system components.

2) Simulation of remote-memory accesses: for each captured remote-memory accessing event, it is necessary to

simulate the access action according to the cache coherence mechnism and page caching information, and by invoking routines of interconnection network simulation, latency of the remote-memory access can be calculated. The simulation method of remote-memory accesses is presented in next subsection.

3) Simulation of target interconnection network and storage system: in NUMA systems, both remote-memory access messages and I/O data are transported via interconnection network, so it is a key factor that influences system performance. By modeling and simulation of interconconnection network and storage system, the latency of remote-memory access and I/O operation can be obtained for each captured event, and finally, the T_{rmem} and T_{IO} of each process can be calculated. This part of work is introduced in detail in subsection-C of this section.

4) Time-axis construction for application processes: the simulator needs to support parallel simulation to achieve performance and scalability. However, with far less processors than the target, the host system must run more processes on each processor than the target. As the result, the execution progress of processes are totally different from the target. To characterize the execution progress of each process in the target machine, it is necessary to construct time-axis for each application process in the target. After occurrence time of each event is fixed in the time-axis, the waiting time of each process, T_{ready}, can be calculated. Detailed introduction to the time-axis construction is given at the end of this section.

B. Capture and Simulation of Remote-memory Accesses

In SimNUMA, instructions of applications are executed on the fly instead of simulated one by one, and to guarantee simulation performance, it is infeasible to check instructions one by one to identify if there is a remote-memory access. So the challenge is how to capture remote-memory accessing events with performance impact as small as possible.

To achieve the above objective, a new method is proposed for our simulator which captures remote-memory accessing events efficiently. The main idea is: simulating physical memory of the target machine by virtual memory of the host system, and using virtual memory protection mechanism of the host operating system to implement the capture of remote-memory accessing event.

As Figure 1 shows, the virtual address space of the host system are divided into multiple sections, each of them corresponds to the physical memory of one processing node in target machine. By setting the size of each section to the physical-memory size of corresponding target node, the virtual address space of each application process are limited into the corresponding section, and according to the virtual memory protection mechanism, an exception will be raised and processed by the operating system if the process accesses memory locations outside of the section. As the result, when an application process in the target system accesses remote-memory located in other nodes, the corresponding process in host system will access an address outside of its virtual address space, and this operation will lead to a page-fault in virtual memory of host operating system. That is, a remote-memory access in target NUMA system corresponds to a page-fault in host system. Consequently, the page-fault, or the remote-memory access, can be captured by the simulator by means of probe-functions added to the kernel of operating systems. After that, the event is simulated and checked if a real remote-memory access occurs or it just hit cached pages, according to the cache coherence mechanism of the target machine and remote-page caching information maintained in the simulator. If a real remote-memory access occurs, the interconnection network simulation module will be invoked, which will return the latency of the data transportation.

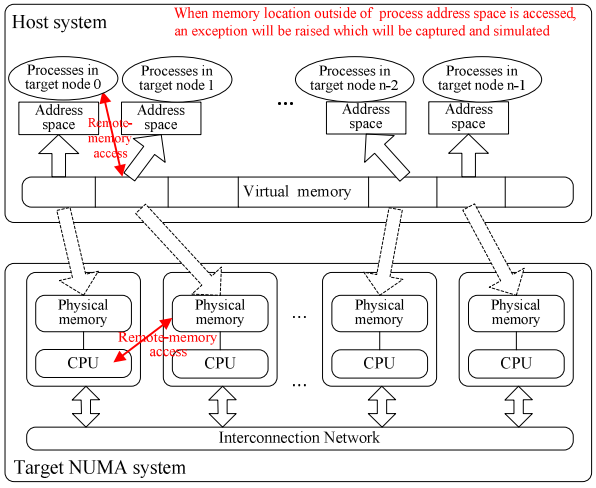

Fig. 1. Method to capture remote-memory accesses

In connecting with the cache coherence mechanism of a real NUMA system, this module utilizes the following simulation scheme to simulate the target machine cache consistency events which are based on the distributed full-map directory and apply MESI write-invalidation protocol. We abstract the directory as an object of which the internal data object is a vector data. The array's unit is a directory entry which needs to record the memory block address, the dirty bit and the node vector in it. Directory object provides interface through which you can add and search directory entry and add its own node information for the directory entry which a certain address is in. Every Remote Cache of the target system corresponds to a cache object. The whole simulation process applies the way of message-driven and there is a cache message queue in CacheSys which is the object of the cache system. The cache system will check this queue and scheduling the corresponding cache object of the cache message to do simulation processing.

C. Simulation of Interconnection Network and Storage System

1) Overview

For each captured events of remote-memory and file accesses, the simulator invokes routines of interconnection network and storage simulation module to calculate the latency of the event in target machine according to the models of target interconnection network. However, modeling of

interconnection networks is rather complex since it involves many factors such as topology and bandwidth of the network, routing algorithms, buffering and queuing, etc. Some other simulators even ignore this part of work or just use constants as the latency.

In SimNUMA, to guarantee simulation performance, we use a relatively detailed performance model of interconnection networks to calculate latencies instead of circuit-level cycle-accurate simulation. The *"relatively detailed performance model"* means that the latency is calculated not just by a simple formula but by considering factors mentioned above including topology and bandwidth of the network, routing algorithms, buffering and queuing, etc.

Currently, the simulator supports two interconnection techniques: Infiniband and Intel QuickPath Interconnect (QPI), and three kinds of topologies: full-connection/full-switch, 2D-Torus and Hypercube. Obviously, other interconnection techniques (e.g. AMD HyperTransport) and network topologies (e.g. 3D-mesh/Torus) can be easily implemented based on current models.

2) Modeling and simulation of interconnection networks

Generally, a remote-memory or file access is accomplished by transporting multiple messages, typically a request and its corresponding reply, with their length depended on the size of data to be transported. Thus the latency of an event can be expressed as:

$$T_{network} = T_{msg\_req} + T_{msg\_reply} \tag{2}$$

At the interface to the network, the request or reply message is split into a group of smaller packets in fixed length. Theses packets are transported to destination node one by one via the interconnection network, and assembled into message at the destination, so the T_{msg} can be calculated as:

$$\begin{aligned} T_{msg} &= T_{source} + T_{network} + T_{destinatio\ n} \\ &= \frac{L_{msg}}{R_{send}} + \left[\underset{pkt \in msg}{Max} (t_{pkt}) - t_{start} \right] + \frac{L_{msg}}{R_{receive}} \end{aligned} \tag{3}$$

Where L_{msg} is the length of message; R_{send} and R_{recv} is the splitting/assembling rate of message into/from packets at source and destination node respectively; t_{start} is the starting time of transmitting; t_{pkt} is the arriving time of packets at destination. Since multiple packets may be transported in the network simultaneously via different routes, and arrive at destination not in sequence, the maximum arriving time is used to count the transport delay of message instead of arriving time of final packet.

After a packet is transmitted into the interconnection network, it is forwarded hop by hop according to the network topology and routing algorithms. To calculate arriving time of packets at destination according to the network model, each packet carries a timestamp which is updated by components of the network along with its transportation. During this process, mainly two types of components are involved in: *NodeController(NC)* and *Link*, where *NC* is the abstraction to the switching component which connects processors of node to other *NCs*.

The other component *Link* connect *NCs* each other to form the interconnection network. The latency of the packet on *Link* can be calculated as:

$$T_{link} = \frac{L_{packet}}{B_{link}} + T_{Link\,\mathrm{Pr}\,op} + T_{flowcontol} \tag{4}$$

Where B_{link} is the uni-directional bandwidth of the channel. $T_{linkProp}$ is the propagation delay of the physical circuits. The meaning of $T_{flowcontrol}$ here is same as that in *NC*. It is needed to note that the coding mechanism of the link should be taken into account, for which could make the number of packets split from a message even more. Also note that the link here is full-duplex that means it can transmit packets bi-directionally at the same time.

On the basis of above analysis, more detailed models can be built according to specific interconnection techniques and topologies, and used to calculate latency of remote-memory or file accesses.

3) Interconnection simulation for Intel Xeon: an example

Figure 2 gives a simple example of interconnection network based on Intel Xeon, in which processers are connected via Intel QPI, and *NodeControllers* are used to connect two-way nodes in the form of a 2-ary 2-cube network.

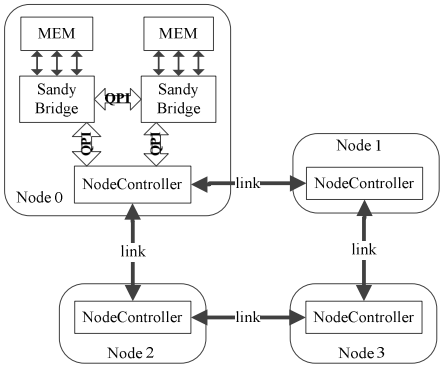

Fig. 2. Example of interconnection network for Intel Xeon 5500.

Suppose that *node 0* intends to read a block of memory located in *node 3*. Firstly, a read-request message containing memory address is generated on *node 0*, then the message is converted into one or more packets in the *NC* of *node 0* with some time delay, after that, packets are simulated to pass through one *link* to the *NC* on *node 2* or *1* and another *link* to *NC* in *node 3*. When arriving at *NC* in *node 3*, timestamp of each packet is updated according to the principles mentioned before. Once the last packet arrives at *node 3*, the overall latency of the request message is calculated with some receive overhead. The latency of corresponding reply message can be calculated in the same way.

4) Simulation of file accesses

The latency of a file access is not only decided by performance of I/O equipment, but also related to internal status of operating system, especially file prefetching and buffering status. Since the host system has the same processor and operating system with the target, it can be assumed that

software-related status in the host is similar to the target, that is, status of file read prefetching and write buffering are similar and unnecessary to simulate. Consequently, only physical read/write accesses to files are needed to be simulated, and in the target machine, these accesses are sent to I/O equipment via interconnection network. So the latency of a file access is composed of two parts: latency in interconnection network and I/O equipment's, in which the former factor can be achieved by invoking interconnection network simulation module, and the latter factor is calculated according to average read/write delay of the target I/O equipment.

D. Time-axis Construction for Processes

The simulator uses the time-axis to characterize execution progress of application processes. The time-axis of a process is composed of various kinds of events that influence execution of the process and their occurrence time, and it can be defined as follows:

In the host system, by capturing concerned events during execution of applications, time-axis of each process in the host can be obtained. The objective of the simulator is to obtain time-axis of each process in the target system, or target time-axis, which is constructed according to the host time-axes of related processes and scheduling policy of the target operating system.

Communications and synchronizations among processes influence execution progress of processes. So in the construction of target time-axis for a process, not only host time-axis of the process but also time-axes of other related processes are needed to be referenced, in order to calculate the waiting time of processes in communications and synchronizations.

In the course of target time-axes construction, the simulator analyzes captured events one by one according to the order of their captured time, and drives forward time-axis of each process on the basis of event type, its occurrence time in the host and returned result of other simulation modules. A logic clock is used to record current position of each time-axis, with the value initially set to 0, the clock increases along with the occurrence time of events in the target machine been fixed one by one. In addition, a communication event queue is maintained for each process in the simulator to match the communication events such as send/receive between processes.

IV. ARCHITECTURE OF THE SIMULATOR

The SimNUMA is currently implemented in Linux system, and its architecture is shown in Figure 3. The system is composed of six modules including event capture, analysis, interconnection network & storage system simulation, control, configuration and statistic & visualization module.

The event capture module runs in kernel of Linux system, and by adding probe-functions to the kernel, the corresponding function of the module is called on occurrence of concerned events including remote-memory access, process scheduling, file access, inter-process communication etc. All of the captured events are transmitted from kernel to the event analysis module which runs in user mode via pre-allocated exchange buffers.

The event analysis module analyzes events one by one, invokes interconnection network and storage system simulation module when necessary, and constructs time-axes in the target system for each application process. The results are written into a log file, which will be used by statistics & visualization module.

The system control module provides a command-line interface to users and supports startup, stop and configuration commands. During startup of the simulator, the configuration module obtains various kinds of parameters about target machine from a configuration file, such as number of processors, memory size, interconnection network parameters such as type, topology, bandwidth, routing policy, etc.

Unlike the modules mentioned above, the statistics & visualization module runs in Windows system, and provides different kinds of statistics and graphs via a GUI interface on the basis of the log file output by the event analysis module.

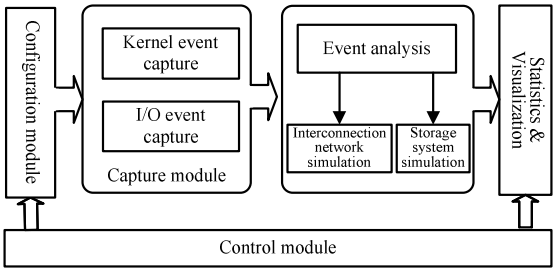

Fig. 3. Architecture of the simulator.

It is needed to note that Linux system support threads in user space and treat threads as processes in its kernel. Under this background, the event capture module of our simulator only captures concerned events of processes. This is also the reason that we use the term *process* to denote concurrent unit of target operating system. However, the simulation methods discussed in previous section can also be applied to thread-based operating systems such as Windows, in which the event capture module can trace and capture events of threads via various kinds of interfaces, such as hook-functions and event tracing (ETW) mechanism.

V. EVALUATION AND ANALYSIS

A. Overview

The simulator is currently running in a server with two-way quad-core Intel Xeon processors. The accuracy and performance of the simulator are tested using benchmark applications from SPEC CPU2006 and SPLASH-2, then the system is used to simulate a series of NUMA-architecture target machines with different number of nodes and interconnection networks.

Configurations of the experimental environment are below, We use the IBM x3650 M3 Rack-mounted Servers as the host machine, with the Intel Xeon E5620, two quad-core, 12GB DDR3 RDIMM memory and three 300G SAS Hard Disks.

And our Operation system is Ubuntu Server Linux 11.04(kernel:2.6.38). The applications we used are bzip2, hmmer, sjeng, sphinx, dealII(SPEC CPU 2006) and LU(SPLASH-2).

B. Accuracy and Performance Test

1) Accuracy Test

Accuracy of the simulator is tested by comparing execution time of benchmark applications in simulated with that in real target machine. Firstly, small-scale target systems with 1 and 2 nodes are modeled in the simulator, then 6 benchmark applications are executed in both of the simulated systems and real target machine. When execute single SPEC CPU program on two nodes, we make two instances of the same program run in two processors. After that, the simulated execution time is compared to the real execution time, and error rate of the simulation are calculated, as shown in Figure 4.

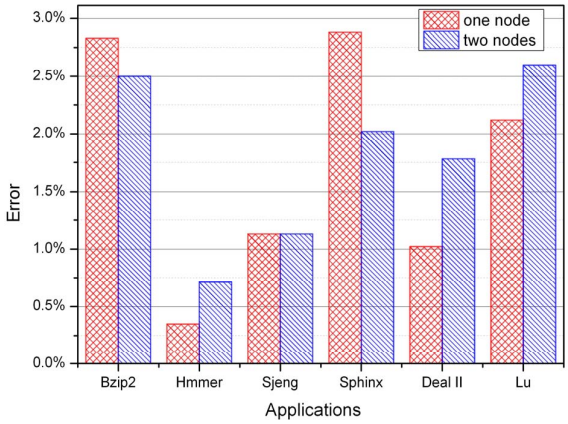

Fig. 4. Accuracy of the simulator.

The real target machines used in accuracy test are the host machine itself without running our simulator. It is needed to note that the host machine used in the test is not traditional SMP but NUMA architecture[22], since the DIMMs of the machine are connected to memory controllers of two Intel Xeon processors separately, leading to different latencies for accessing local and remote memory, according to the product specifications from the vendor, the latency to access remote memory is almost 75% higher than local memory access[22].

Then in order to tell the accuracy difference compared to Gem5, here give the test result on 16, 32 and 64 nodes as shown in Figure 5.

Figure 5 shows the margin of error of simulator on 16, 32 and 64 nodes. It's obvious from the histogram that, the larger the node size is, the greater the error is. As all the margin of error is bigger than 10%, there is still a gap between Gem5. However, Gem5 has some differences of indicator framework between our simulator, and our simulator focuses on the performance of NUMA computer, so the loss of accuracy is inevitable. Finally, to see whether a simulator is valuable, is not depends on if the absolute numerical simulation results are close to the target system, but depends on whether the relative trends on different configurations is right. So the accuracy of our simulator also can be accepted.

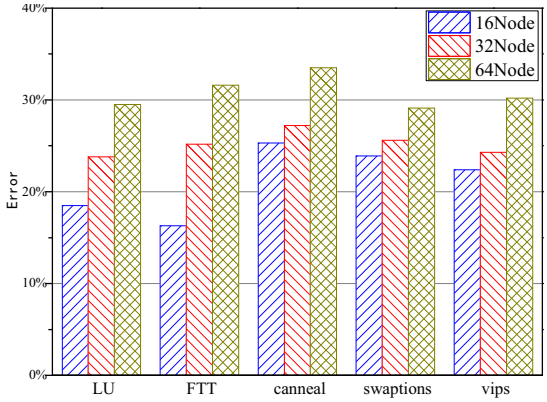

Fig. 5. Accuracy of the simulator(16,32,64nodes)

2) Performance Test

Both slowdown factor and parallel simulation speedup of the simulator are tested to verify its performance and scalability, in which the slowdown factor is the ratio between simulation time of applications and their actual execution time without the simulator, and can reflect the overhead of the simulation, while the speedup reflects effects of parallel simulation, i.e. scalability of the simulator.

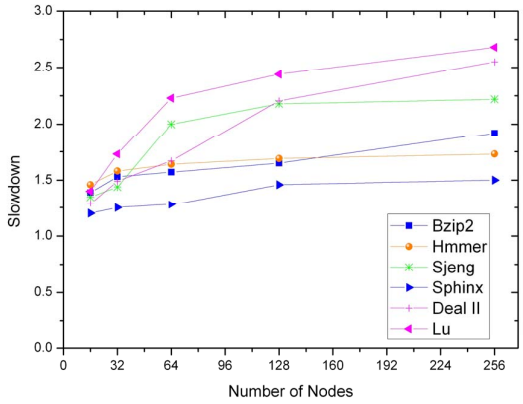

Fig. 6. Slowdown of the simulator.

Figure 6 shows slowdown results of SimNUMA in simulating target machines with different number of nodes. According to the results, the smaller the value is, the higher performance the simulator shows. And the maximum slowdown is only 2.68 times, as a contrast, slowdown factor of RSIM for the same application LU varies from 15 to 7100 times under different simulation modes, as for general-purpose simulators, their slowdown factors are generally more than 1,000 times, sometime even more than 10,000 times.

By simulating the same target machine using different number of processor cores, speedup of the simulator can be calculated, as shown in Figure 7.

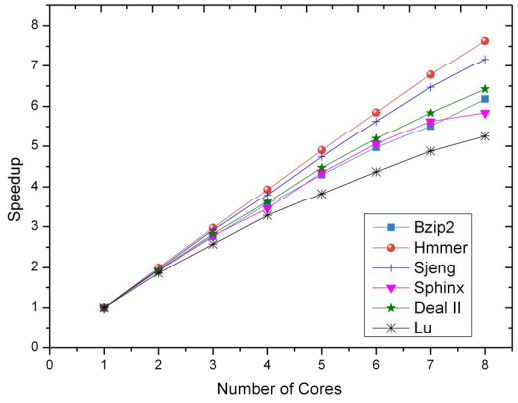

Fig. 7. Speedup of the simulator

3) Interconnection Network Simulation Test

In order to find the function of the interconnection network module and the relationship between target system performance and the internet type, we have the follow experiments with different internet types of five test programs. Because of the topological characteristics, here we select 64 nodes for the target node, and every node configures a processor.

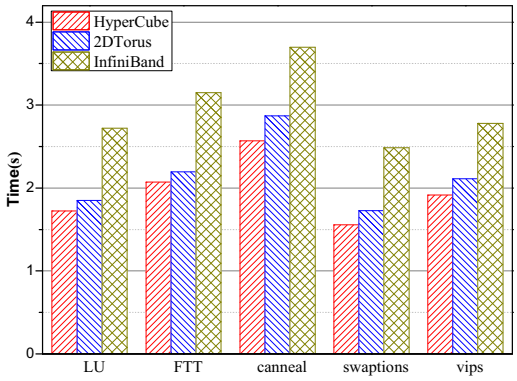

Fig. 8. Test results of interconnection network module.

Figure 8 shows all the performance of the five programs on three different internet types with the interconnection network module. As shown in the figure, the performance of HyperCube with 64 nodes is better than that of 2D-Torus when the other configration is same, which is consistent with the difference between their topology characteristic such as their network radius.

Figure 9 shows the average time delay on different link speed and different scales of problems. The results show that as the link speed increases, there is a decrease tendency of the end delay of the interconnection network, especially when the bandwidth is low. But when the bandwidth reach a threshold number (in this experiment, the number is 50Gb/s), the increasing of the bandwidth has fewer influences on the time delay. The reason why time delay change much is because the main contradiction occupying the communication delay is the fixed delay between each two switching equipments, and the bandwidth only determined the transfer delay when the data packets exchanged between switching equipments,

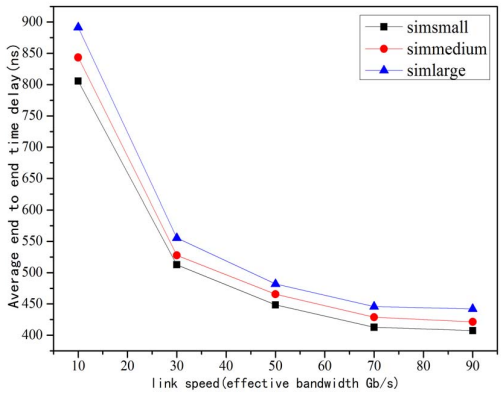

Fig. 9. Time delay on different link speed and different scales of problems

C. Simulation Experiments

On the basis of accuracy and performance test, the simulator is used in a series of experiments to simulate target machines with different number of nodes and interconnection networks. In the experiments, node number of the target machine varies from 16 to 256 with each node equipped with one quad-core Intel Xeon processors; all of the nodes are connected with each other using Infiniband or QPI; the topology of the interconnection network are full-switch for Infiniband, 2D-Torus and Hypercube for QPI; bandwidth of Infiniband and QPI are set to 40Gb/s and 80Gb/s respectively.

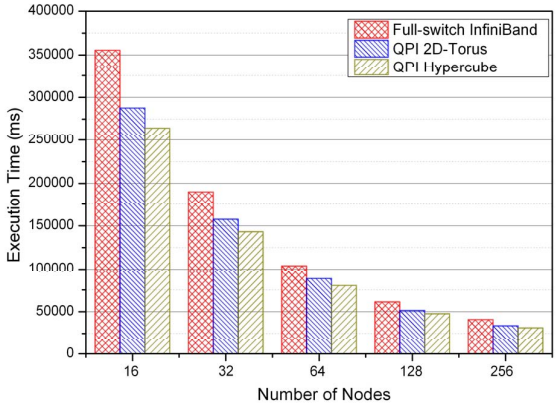

Fig. 10. Simulated excution time in different target machines

Figure 10 shows execution time of applications for different kinds of target machines. From the figure, with the number of nodes increases, execution time of the same application decreases at the same time, while the hypercube network with QPI shows the best performance, due to its smaller network diameter and average distance. It is needed to note that although the performance of full-switched Infiniband network is relatively lower than the others, infiniband-based machines can be easily scale up/down in modular, and can meet requirements of some kind of users.

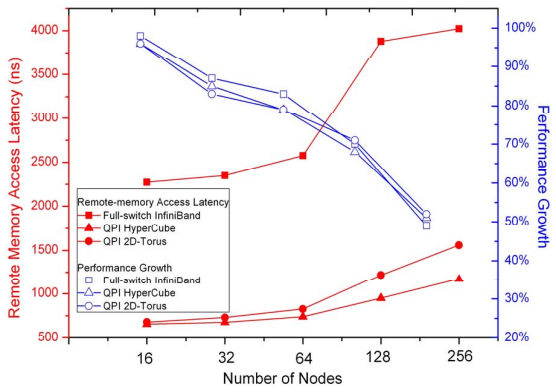

Fig. 11. Averagy remote-memory access latency and performance growth rate.

The relationship between remote-memory access latency and system performance is shown in Figure 11. As shown in the figure, although the system performance increases along with the increase of node number, the growth rate becomes slower and slower. For example, when the number of nodes increases from 128 to 256, the target machine only achieves 51% of performance growth. One of the reason is that the more nodes the system has, the larger the interconnection network is, as the result, the longer the average remote-memory access latency is, as shown in Figure 11.

VI. CONCLUSION

Non-uniform memory access (NUMA) architecture is widely used in high-end servers and computing systems due to its scalability. In recent years, the number of processor cores in NUMA systems increases rapidly with the development of multi-core processors. Along with the growing of system scales, simulation of NUMA systems becomes a challenge to traditional general-purpose simulators due to their low simulation performance.

This paper presents SimNUMA, an execution-driven full-system simulator dedicated for NUMA systems. The main characteristics of the SimNUMA are, it uses the same type of processor with the target machine in the host system to avoid complex and inefficient modeling and simulation of micro-architecture and instruction-set for commercial available processors, on the basis of that, a new method to capture and simulate remote-memory accesses efficiently is proposed, in addition, parallel simulation is supported to achieve scalability. Evaluation results show the simulator achieves satisfied performance and scalability.

ACKNOWLEDGMENT

This work is supported by National Science Foundation of China under grant No. 61073011, 61133004 and National Hi-tech R&D program (863 program) of China under grant No. 20 12AA01A302.

REFERENCES

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

[2] M. Rosenblum, S. Herrod, E. Witchel, A. Gupta, “Complete computer simulation: the SimOS approach,” Parallel and Distributed Technology, 1995, 3(4):35-43.

[3] P. S. Magnusson, M. Christensson, J. Eskilson, et al. “Simics: a full system simulation platform,” IEEE Computer, 2002, 35(2):50-58.

[4] M. Martin, D. J. Sorin, B. M. Beckmann, et al. “Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset,” ACM SIGARCH Computer Architecture News, 2005, 33(4): 92-99.

[5] N. L. Binkert, R. G. Dreslinski, L.R. Hsu, et al. “The M5 simulator: modeling networked systems,” IEEE Micro, 2006, 26 (4) : 52-60.

[6] C. J. Hughes, V. S. Pai, P. Ranganathan, S. V. Adve, “RSIM: Simulating Shared-Memory Multiproeessors with ILP Proeessors,” IEEE Computer, 2002.35(2):pp. 40-49.

[7] V. S. Pai, P. Ranganathan, and S. V. Adve. “RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors,” Proc. of the Third Workshop on Computer Architecture Education, February 1997.

[8] V. Puente, J.A. Gregorio, R. Beivide, “SICOSYS: An Integrated Framework for studying Interconnection Network Performance in Multiprocessor Systems,” Proc. of Parallel, Distributed and Network-based Processing(PDP) , 2002: pp: 15-22.

[9] J. Tao, M. Schulz, and W. Karl, “Simulation as a tool for optimizing memory accesses on NUMA machines,” Performance Evaluation, 2005, 60(1-4): 31-50.

[10] A.T. Nguyen, M. Michael, A. Sharma, J. Torrellas, “The Augmint Multiprocessor Simulation Toolkit for Intel ×86 architectures”, Proc. 1996 International Conference on Computer Design, IEEE Computer Society, October 1996, pp. 486–491

[11] Mc.Collin, V.Jeffrey, “Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms,” Proc. IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS), 2010, pp:87-96.

[12] K.C. Kandalla, H. Subramoni, G. Santhanaraman, M.J. Koop, and D.K. Panda, “Designing multi-leader-based Allgather algorithms for multi-core clusters,” Proc. IEEE International Parallel and Distributed Processing Symposium(IPDPS), 2009.

[13] B. Falsafi, D.A. Wood, “Scheduling communication on an SMP node parallel machine,” Proc. Third IEEE Symposium on High-Performance Computer Architecture(HPCA), 1997, pp:128–138.

[14] A.R. Mamidala L.Chai H.Jin, D.K.Panda, “Efficient SMP-Aware MPI-Level Broadcast over InfiniBand's Hardware Multicast,” Proc. IEEE International Parallel and Distributed Processing Symposium(IPDPS), 2006.

[15] S. Hari, L. Ping, S. Sayantan, P. Dhabaleswar K, “Improving application performance and predictability using multiple virtual lanes in modern multi-core InfiniBand clusters,” Proc. International Conference on Parallel Processing(ICPP), 2010:pp: 462-471.

[16] J.Dean, S.Ghemawat. “MapReduce: Simplied Data Processing on Large Clusters,” Proc. Symposium on Operating System Design and Implementation(OSDI) 2004, pp:137-150.

[17] L. Zhang, L. Cheng and R. Boutaba, “Cloud computing: state-of-theart and research challenges,” Journal of Internet Services and Applications, Springer, 1(1): 7-18, 2010

[18] B.Anant, N,Azad, “Interconnect network analysis of many-core chips,” IEEE Transactions on Electron Devices, 2011,58(9):2831-2837.

[19] Wang Huandong, Gao Xiang, Chen Yunji, Hu Weiwu, “Interconnection of Godson-3 multi-core processor,” Computer Research and Development in Chinese,2010, 45(12):pp:2001-2010

[20] S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” Proc. International Symposium on Computer Architecture(ISCA),1995.

[21] Phansalkar, A., Joshi, A., and John, L. K. “Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite,” Proc. International Symposium on Computer Architecture(ISCA), 2007.

[22] IBM, “Product Guide of IBM System x3650 M3,” IBM Corporation, 2011.