HPPNetSim: A Parallel Simulation of Large-scale Interconnection Networks

Zheng Cao1,2,3, Jianwei Xu1,2,3, Mingyu Chen1,2, Gui Zheng1,2,3, Huiwei Lv1,2,3, Ninghui Sun1,2

1Institute of Computing Technology, Chinese Academy of Sciences, Beijing
2Key Laboratory of Computer System and Architecture, Chinese Academy of Sciences, Beijing
3Graduate University of Chinese Academy of Sciences, Beijing
{cz, xjw, cmy, zhenggui, lvhuiwei, snh}@ncic.ac.cn

Keywords: Parallel discrete event simulation, conservative approach, large-scale interconnection network, super-linear speedup

Abstract

As the scale of parallel machine grows, communication network is playing more important role than ever before. Communication affects not only execution time, but also scalability of parallel applications. Parallel interconnection network simulator is a suitable tool to study large-scale interconnection networks. However, simulating packet level communication on detailed cycle-to-cycle network models is a really challenge work. We implement a kernel-based parallel simulator HPPNetSim to solve problems. Optimistic PDES mechanism needs huge memory consumption of saving simulation entities' states in large-scale simulations, so we chose conservative synchronization approach. Simulation kernel and network models are all carefully designed. To accelerate process of simulation, optimizations are introduced, such as block/unblock synchronization, load balancing, dynamic look-ahead generation, and etc. Simulation examples and performance results show that both high accuracy and good performance are obtained in HPPNetSim. It achieves speedup of 19.8 for 32 processing nodes when simulating 36-port 3-tree fat-tree network.

1. Introduction

To meet the challenge of Petaflops computing, ultra-large scale parallel machines have been released. According to Top500 list released in June 2008, more than 30 machines are built with more than 10K processor cores. Furthermore, scale of HPC systems will keep growing with the increasing demand of computing capability. As a result, more and more time will be spent in the process of communication. In that case, not only applications’ execution time, but also system’s scalability will be affected. Therefore, to deliver high bandwidth and low latency, high performance large-scale interconnection network must be carefully designed.

Before bringing a proposed network into implementation, both network entities and network architecture design should be evaluated first. Compared with mathematical models, simulator is the suitable evaluation tool, especially for ultra-large networks. Many interconnection network simulators have been developed to carry out evaluation at low level (such as multiplex, queue, and crossbar) and guide both micro-architecture and network design. But many of them [1-5] are written with sequential codes, as a result, they will reach practical limits in two aspects: a) A sequential simulator is too slow to get the result in expected time, since it exploits the computing capacity of no more than one processor. b) A sequential simulator cannot satisfy the requirement of a large-scale network with huge physical memory. Therefore, parallel network simulator is essential to evaluate large/ultra-large networks.

Several parallel simulators have been developed for interconnection networks. [6] and [7] describe parallel simulation of k-ary n-cube interconnection networks. [8], [9] and [10] focus their effort on multistage interconnection networks (MIN). However, these works are limited to a subset of network topologies and the network scale is limited. Furthermore, since most of them were developed in the mid-1990s, many new techniques in communication are not involved. BigNetSim [11] can simulate both direct and indirect networks in large scale and has achieved good parallel performance. But large memory consumption is still a problem and it is not widely used because of the tight-couple with CHARM++.

In this paper, a parallel interconnection network simulator HPPNetSim is developed. It simulates packet level communication on detailed cycle-to-cycle network models for large/ultra-large scale network. Simulating large scale network in hardware level is a challenge work. However, by designing parallel simulation kernel and network models carefully, it achieves speedup of 19.8 for 32 processing nodes when simulating 36-port 3-tree fat-tree network, including 11,664 nodes and 1,620 switches.

The rest of this paper is organized as follows: Section 2 introduces simulator architecture, which describes simulation kernel and network models in details. Section 3 takes HPPNet [12] as a case to give a verification of the simulator. Section 4 shows performance results of simulator. Finally, section 5 gives the conclusion and our future works.

2. Simulator Architecture

HPPNetSim is designed with kernel-based architecture. As shown in Figure 1, SimK plays the role of simulation...
kernel, which takes in charge of communicating and synchronization among simulating elements. Models, such as routers and switches, are attached to the SimK to do cycle-to-cycle simulation. In trace-driven mode, HPPNetSim itself performs network simulation by using artificial traffic or trace files as its input. In execution-driven mode, HPPNetSim is integrated with HPPSim (a full system simulator) to get real-time traffic generated by applications.

2.1. SimK

SimK is a general-purpose conservative synchronized PDES environment. It is written in C language and provides infrastructure modules for parallel simulation, including deployment, scheduling, communication, buffer management, and time synchronization. Figure 2 shows its architecture.

![SimK architecture](image)

**Figure 2** SimK architecture

In SimK, simulation modules of target system, such as switch modules, are described as logical element (LE). Initialization and deployment of LEs are managed by the deployment module at initial stage. During simulation, scheduling module puts each LE into proper running queues, from which the execution module repeatedly takes out LEs and executes the callback functions to do real simulation. Correctness and efficiency of communication between LEs are guaranteed by the communication module. Synchronization module, built upon the execution and communication module, is dedicated to maintain the time synchronization of all LEs.

2.1.1. Task allocation and deployment

In conventional simulators [13-16], modules cooperate with each other by direct function calls. Such approach is not suitable for parallel simulators where all modules run concurrently. It requires all functions to be reentrant which makes the simulator with poor scalability and impose extra burden to developer. In SimK, modules and LEs communicate with each other through message passing.

LE is the basic unit for task allocation. At initial stage, all LEs are mapped to host processors as balanced as possible. As we know, LEs on same processor can share private cache and have low communication overhead, so the mapping process falls into a classical graph partition category. It can be described as follows: given a graph $G$ with $n$ weighted vertices and $m$ weighted edges, we need to divide vertices into $p$ sets under following rules: 1) sum of vertexes’ weight inside a set is close to sum values in other sets; 2) sum of edges’ weight between sets is as minimum as possible. This problem is known to be NP-complete [17], but there are some heuristic approximate solutions. An outstanding graph partitioning package Chaco [18], which is developed by Sandia National Lab, combines many techniques involved in geometric algorithm, combinatorial algorithm and spectral methods, etc. Experiments show that Chaco satisfies our needs well, so it is used in SimK.

2.1.2. User level schedule

It is observed that each LE had to synchronize with other LEs every dozens of microseconds when performing a simulation. If a LE was delayed for a long time, all other LEs would be stalled directly or indirectly. Using Linux default thread scheduling granularity, experiments show that when two threads in SimK are running on one processor, more than 99% of the time is wasted on waiting.

To solve the problem, a user level scheduling scheme is proposed in SimK. Firstly, only one process is created on one host node, and several threads are spawned within the process. The number of those threads is no more than the number of processors of the node. Thus race between multithreads running on one processor is avoided. Secondly, LEs dispatched to threads evenly, and scheduling of them is restrained in thread scope. To go further, SimK dedicates every host processor to a particular thread using CPU affinity system calls which were introduced in Linux kernel 2.5.8. By doing this, when a thread ceases its execution on one processor and then continues on another one, cache and TLB replacement penalty are avoided.

In Linux there is a state descriptor for each process to describe what is happening to the process. Similarly, LE sets a state flag to describe its current mode. According to the flag, the scheduler decides whether the LE can be executed or not at next turn.

Load balancing is also implemented in schedule module. Existing dynamic load balancing algorithms usually separate decision progress and migration progress [19-20], which are not well fit into sub-millisecond granularity migration. The reason is that between decision and migration gap, the load view of system might have changed. Inspired by work stealing algorithm [21] used in MIMD-style parallel computers, a cooperated migration mechanism is proposed, which combines the merit of work sharing and work stealing, and takes into account cache effect on multi-processor system. Our mechanism separates `run_list` and `mig_list` for normal execution and migration.
Inside a thread, LEs are scheduled and steal as follows:

1. All ready LEs are first put into run_list. If the affinity flag of the LE indicates it has just been migrated, it is put to tail, otherwise to head.
2. Checks the length of run_list, if it has beyond a threshold, excessive LEs are moved to mig_list.
3. Checks local run_list, if it is not empty, LEs in it are sent to execute, and LEs in local mig_list are moved to local run_list. Otherwise, it tries to steal LEs from remote mig_list.

Using this approach, conflict between normal execution and migration, and the loop stealing introduced by original approach are eliminated.

To remove the cache false sharing introduced by run_list and mig_list, the run_list and mig_list are put to separate cache lines. Experiments show that this approach brings down the cache miss ratio which mig_list access occupied from more than 40% to less than 8%.

2.1.3 Block/unblock based synchronization

In SimK, conservative PDES mechanism is chosen, which can avoid huge memory consumption of saving simulation entities’ states, especially in large-scale simulations. In general PDES frameworks [22-24], all simulation entities are scheduled fairly, assuming that each component has work to do at each turn. However, this assumption is not always right in practical. In a large scale simulation, there are many undeterminate factors that will cause uneven run time of each thread. If LEs are scheduled and run at each turn unconditionally, there will be often the case that a short-run-time LE runs without doing any essential work because it has to wait for other long-run-time LEs to catch up. Such case is called False Run (FR) in SimK. FR will waste system resources. Even worse, if the short-run-time LE is not carefully written and increases its local timestamp at each run, synchronization of the system will be violated.

Here, FR is avoided based on block/unblock mechanism. LEs can notify SimK to set its state to blocking with an API function provided by SimK. Based on different blocking reasons, such as waiting event or synchronization constraint, different blocking state will be used. SimK will bring the blocked LE into unblocking until the waiting event arrives or the synchronization constraint is broken.

2.1.4 High efficiency communication

Experiments show that there is nearly one communication every 1us when SimK runs on one processor (CPU frequency is 2.4GHz). When running on four processors, communication occurs nearly every 0.5us. In this case, a high efficient communication mechanism for SimK is crucial. SimK implements an asynchronous zero-copy communication lib. Memory space is shared by all LEs, so sender directly put buffer address into the receiver's input queue, avoiding buffer copy and relay overhead in general communication library.

2.2. Simulation Models

HPPNetSim is designed to simulate large/ultra-large interconnection networks and study the communication behavior of parallel applications. In the full system simulator HPPSim, network is abstracted as a single black-box switch, which simulates communication with a constant delay. In execution-driven mode, HPPNetSim replaces the black-box switch and does the contention-based network simulation.

Generally, an interconnection network is character with topology, routing mechanics, switching mode, flow control mechanism, and micro-architecture of network entities. Features of target networks we simulate are listed below.

1) **Topology:** As the first version, only regular multistage interconnection network is well supported. Using automatic topology generating tool, regular MINs can be initialized by setting some parameters. Topology generating tools for other topologies are works at next stage.

2) **Routing:** For fat-tree topology, both oblivious and deterministic routing algorithms are supported, but for direct networks, only deterministic dimension-ordered routing is implemented.

3) **Switching:** Virtual cut-through (VCT) achieves a higher throughput and lower latency than wormhole switching. Moreover, the traditional disadvantages of VCT switching, as buffer requirements and packetizing overhead, disappear with technology development of VLSI. Thus, VCT is more preferred in HPC networks [25] [26]. So only VCT switching is chosen.

4) **Flow Control:** Credit-based flow control mechanism is implemented.

5) **Network accelerator:** In addition to the fundamental aspects defined above, network-based barrier operation is also simulated. As we all know, time spent in barrier grows with the scale of system. Several systems [27-29] implement dedicated or embedded barrier network to accelerate barrier operation. To study the behavior of barrier and its effect to application performance, HPPNetSim simulates an embedded barrier network, which implements tree-based barrier protocol in NICs and switches.

Micro-architectures of network elements are simulated in details. As shown in Figure 3, NIC model simulates both traditional NIC in indirect network and router in direct networks. Different latency for upstream and downstream DMA operation is modeled. In execution-driven mode, NIC/router gets downstream DMA requests from north bridge model, and translates these requests into network packet format. In trace-driven mode, network packets are either generated by a built-in trace generator, which generate artificial traffic patterns, such as bit complement, transpose, perfect shuffle, hot region and uniform random...
traffic, or read from trace files.

In HPPNetSim, as shown in Figure 4, input buffered crossbar-based switch is modeled. Incoming packets are dispatched to receive buffers based on their virtual channel number. Two kinds of virtual channel allocating methods are supported, one is round-robin, and the other one is for Virtual Output Queue [30], however, in both theme, barrier packets use a specific receive buffer. Besides, a barrier module is implemented, so that barrier packets can be processed with unicast packets in parallel. Arbitrators in crossbar module which can be configured as priority-based, round-robin or matrix arbiter, decide which virtual channel can send packet out. At output port, barrier and unicast packets are arbitrated by priority-based output arbitrators.

In network models, fine-grained block/unblock mechanism is implemented. In three situations, network entities set themselves into blocking state. The first one is waiting for flow control packets when credit is not enough for sending packet out. The second one is that port with smallest timestamp does not have packets to send out. The last one is that packet can’t be sent out because of conservative synchronization constraint. By setting blocking state precisely, simulation performance is greatly improved.

3. Example

In this section, we take HPPNet as an example to give a verification of the simulator. Using HPPNetSim, behavior of HPP Switch and HPPNet are studied.

3.1. HPPNet

HPPNet is the interconnection network in Dawning 5000A project. Dawning 5000A is one of R&D projects of Chinese 863 high-tech program. Target of this project is to build a 100Tflops HPC system, and a prototype for peta-scale system. As shown in Figure 6, HPPNet uses 3-level fat-tree topology with 16-port switches and implements an embedded tree-based barrier network.

Figure 7 shows micro-architecture of the 16-port switch named HPP Switch [12]. Features of HPP Switch are listed below:

1) Credit-based flow control;
2) Virtual Cut Through switching and input-buffered...
architecture;
3) Supports unicast, multicast and barrier operation;
4) Three virtual channels for unicast, one hybrid virtual channel for multicast/unicast, one virtual channel for barrier. Each unicast virtual channel uses a buffer with 4KByte, while barrier only uses buffer with 128Byte;
5) Matrix arbiter is used for arbitration.

Figure 7 Micro-architecture of HPP Switch

3.2. Simulation of HPP Switch

In the experiment, all parameters are set to values in HPP Switch. Besides, uniform random traffic pattern is chosen. Without particular illustration, unicast packets with length of 256Byte are used. Based on the detailed switch model, accurate results are obtained, which are very close to results from RTL simulator [12].

As an input buffered switch, head-of-line (HOL) blocking has greatly influence on the throughput. Figure 8 shows the unicast throughput with different number of VCs. Compared with 1 VC, throughput with 2 VC can increase by a factor of 1.27, 1.37 times with 3 VC, and 1.41 times with 4 VC. Based on a comprehensive consideration of throughput and implementation cost, 3 VCs configuration is a cost-effective choice.

Figure 8 HPPSwitch: Throughputs vs. Number of VCs

Figure 9 shows switch throughput with variation of packet length. Compared with length of 256Byte, throughput only improved 1.4% than using 1024Byte. So setting MTU to 256Byte is suitable for HPP Switch. However, HPP Switch still set MTU to 1024Byte with other consideration.

Figure 9 HPPSwitch: Throughput vs. Unicast packet length

Because embedded barrier network shares network entities with communication network, barrier operation will be affected by unicast packets. Figure 10 shows that barrier delay is relative with length of unicast packet. As packet length grows, barrier delay increases in linear. In Figure 10, the more straight line is measured with RTL simulator, while another line is from HPPNetSim. The two close lines demonstrate that simulation to barrier operation is with high accuracy.

Figure 10 HPPSwitch: Barrier delay vs. Unicast packet length

3.3. Simulation of HPPNet

In this section, some primary tests for HPPNet are carried out. Artificial traffics are used to analyze the performance of HPPNet, such as bit-complement, transpose, perfect-shuffle, hot region (25% of traffic to 12.5% of nodes) and uniform random traffic. Deterministic routing mechanics is chosen for the experiment.

In Figure 11 (a), under uniform random traffic, throughput of HPPNet can achieve more than 70%. However, in Figure 11 (b), under non-uniform traffics (such as bit-complement, shuffle, hot-region and transpose), throughput can get 25% at most.
Experiments show that traffic patterns and unicast packets have effect on barrier performance. In Figure 12(a), because of the high priority, barrier operations are finished in low latency, which is very close to the value of unicast latency at saturation point (See Figure 11). In Figure 12(b), barrier delay still increases linearly with unicast packet length and the worst value is gained under bit-complement traffic. Meanwhile, barrier operation only leads to little degradation of unicast throughput.

![Figure 12](image)

### 4. Performance Evaluation

Performance of HPPNetSim is evaluated by simulating fat-tree networks ranging in size from 686 nodes (14-port 3-tree) to 11,664 nodes (36-port 3-tree) under uniform distribution of random traffic. This pattern was chosen because it would more closely resemble the behavior of arbitrary collection of applications than the other standard patterns. Network parameters were selected to simulate the fat-tree network of HPPNet. During simulation, 64 bytes was chosen as packet’s length and each run simulated 100ms of wall clock. In each run, more than 150 million packets were transmitted. As shown in Table 1, experiment platform is Tiankuo A950r-F Server made by Dawning Information Industry Co., Ltd.

#### Table 1 Experiment Environment

<table>
<thead>
<tr>
<th>Node name</th>
<th>Tiankuo A950r-F</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU number</td>
<td>8</td>
</tr>
<tr>
<td>CPU type</td>
<td>Quad-Core AMD Opteron: 2.2GHZ</td>
</tr>
<tr>
<td>L2 CACHE size</td>
<td>16M (512K×4×8) bytes</td>
</tr>
<tr>
<td>GCC Version</td>
<td>4.1.2</td>
</tr>
<tr>
<td>Compile option</td>
<td>O2</td>
</tr>
</tbody>
</table>

In trace-driven mode, simulations fit in memory comfortably. Experiments show that memory consumption grows linearly with network scale. When simulating 11,664 nodes, 2.2GByte memory is consumed, while scale of 65,536 needs 9GByte memory.

Figure 13 shows a plot of speedup relative to sequential simulation time. HPPNetSim scales well as the number of processors increase. When network scale is smaller than 3456 nodes, super-linear is achieved. Experiments show that the super-linear speedup mainly comes from last level cache effect. Normally, if more threads are employed, more cache will be utilized, and the capacity miss will be decreased. When the benefits of reduced cache miss surpass the overhead of the parallelization, the super-linear speedup emerges. When network scale becomes larger, cache effect gradually disappears and speedup is only provided by parallelization. However, when simulating network with 11,664 nodes, speedup of 19.8 for 32 processing nodes is still obtained.

![Figure 13](image)

### 5. Conclusion and Feature Work

Simulating packet level communication on detailed cycle-to-cycle network models is a really challenge work. Several efforts are made in this paper to accelerate such simulation. Firstly, we design a conservative PDES simulation kernel SimK, which implements functions of user level scheduling, load balancing, block/unblock synchronization, high efficiency communication, and etc, to support large-scale simulation effectively. Secondly, we design detailed network models and use dynamic look-ahead value to achieve high accuracy and good performance. In trace-driven mode, simulating network with
11,664 nodes consumes 2.2GByte memory, so that a general server can be used to do large-scale network simulation. Experiments show that HPPNetSim is with good scalability. When simulating network with 11,664 nodes, speedup of 19.8 for 32 processing nodes is achieved. For network with smaller scale, even super-linear speedup is obtained.

However, there are still a lot of works to do. Firstly, to make simulation of direct networks more convenient, automatic tools for generating topology and more routing algorithms will be implemented. Secondly, in execution-drive mode, simulation time is still so unacceptable that only networks with very small scale can be simulated. There are two reasons. One is that the cycle accurate large-scale system simulator itself is slow. The other one is that, frequency of communication is very low with real applications, most of the time, network models only advance their timestamp by look-ahead value given by North Bridge, which is too small to promote the simulation fast. Thus, learning from BigNetSim, we are planning to use trace files with dependency information to drive the simulator. Finally, further performance evaluation for large-scale direct networks will be undertaken.

References


