HiNetSim: A Parallel Simulator for Large-Scale Hierarchical Direct Networks

Zhiguo Fan, Zheng Cao, Yong Su, Xiaoli Liu, Zhan Wang, Xiaobing Liu, Dawei Zang, Xuejun An

HiNetSim: A parallel simulator for large-scale hierarchical direct networks*

Zhiguo Fan[12], Zheng Cao[1†], Yong Su[12], Xiaoli Liu[1], Zhan Wang[12], Xiaobing Liu[1], Dawei Zang[12], Xuejun An[1]
[1] Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences
{fanzhiguo, cz, suyong, liuxiaoli, wangzhan, liuxiaobing, zangdawei, axj}@ncic.ac.cn

Abstract. As the scale of high performance computer keeps increasing, the hierarchical high dimension direct network, such as Cray Dragonfly and K computer 6D mesh, becomes commonly used. In such architecture, the variety of topologies in each hierarchy leads to the complexity of topology and routing algorithm. Facing to the high complexity and scalability, parallel network simulator is the suitable platform to design network architecture efficiently and study its performance. We design and implement a packet-level parallel network simulator HiNetSim which can achieve both high accuracy and efficiency. In addition, the simulator provides flexible interfaces and configuration files for establishing hierarchical topologies and implementing routing algorithms. As a demonstration of HiNetSim, studies on flattened butterfly, 4D Torus and a proposed hierarchical network are given. Evaluation shows that HiNetSim achieves linear parallel speedup and is capable of simulating the network with tens of thousands nodes.

1 Introduction

Today, the scale of interconnection network is keeping increasing and is expecting to hold hundreds of thousands of nodes [1]. To balance the scalability and performance, the hierarchical direct network architecture corresponding to the communication locality is widely used. In the hierarchical network, different hierarchies of the network often possesses different switching capacities, such as IBM BlueGene/Q 5D torus [2, 3] K tofu nested 6D torus [4] and Cray dragonfly [5]. Facing with different switching capacities, the network topology varies between different network hierarchies. In this case, the full system network architecture becomes more complicated and the fully adaptive routing algorithm is hard to design.

Network simulator is the most commonly used platform to perform the network evaluation. However, to study the complex hierarchical network efficiently, a network simulator must fulfill the following requirements.

1) *Scalability:* the network simulator must be capable of simulating large scale network. Only when the network scale is large enough, can many performance issues such as load balancing and network congestion show up.

* Supported by the National Natural Science Foundation of China under Grant No.61100014
† Corresponding author

2) *Accuracy:* the microarchitecture of network devices should also be simulated in detail, since they have great effect on the full system performance. For example, the number of virtual channels is a key factor for both the network performance and deadlock-free routing algorithm.
3) *Flexibility:* user interfaces should be friendly enough to quickly implement new architectures (topology and corresponding routing algorithm) for different hierarchies.
4) *Performance:* the simulation of the large scale network should be finished in a reasonable time.

However, most simulators are written with sequential codes and cannot meet the requirement of scalability and performance, such as BookSim [6], Xmulator [7], CINSim [8], MINSimulate [9] and INSEE [10]. Some parallel simulators, such as NSIM [11], simuRed [12] and topaz [13], are limited to a subset of network topologies and cannot meet the requirement of flexibility. BigNetSim [14] can perform efficient parallel simulation. However, it is designed with a new language Charm++ which introduces a long learning curve for users to develop new topologies and routing algorithms. The Network Simulator (ns) series [15] and DCNSim [16] are focus on internet system and lack of the micro-architecture details.

In this paper, we design and implement a parallel cycle-accurate network simulator HiNetSim to perform studies on the complex hierarchical network architecture. HiNetSim is written in C/C++ and uses a kernel-based framework. The simulation kernel SimK [17] is in charge of the functions needed by the PDES (parallel discrete event simulation) including the synchronization and communication mechanism, simulation task scheduling, and memory management. In this case, users can focus only on the simulation of network behavior which greatly reduces the developing time of new network models.

As the building block of hierarchical network, basic network architectures, such as fat-tree, all-to-all, *n*D torus, *n*D mesh, flattened butterfly and etc., have been implemented in HiNetSim. In addition, HiNetSim provides flexible user interfaces and configuration files to customize topologies and routing algorithms.

The reset of paper is organized as follows: Section 2 discusses key issues of designing HiNetSim; Section 3 describes the architecture and implementation of HiNetSim; Section 4 demonstrates the ability of HiNetSim by simulating some example networks. Section 5 shows performance result of HiNetSim. Section 6 gives the conclusion.

2 Key Issues and Design

2.1 Parallelism

In HiNetSim, each network component, such as NIC and switch, is defined as one *LE* (Logic Element). *LE* is the basic element for parallel simulation and scheduling.

Timing synchronization between *LEs* is the key issue regarding the correctness of simulation function. We adopted PDES (Parallel Discrete Event Simulation) mechanism [18] a parallel distributed synchronization mechanism to solve this.

There are two PDES mechanisms: conservative and optimistic. In the conservative model, events with later time stamp can only be processed after the earlier ones. In the optimistic one, events with later time stamp may be processed in advance. If an earlier time stamp arrives, a rollback mechanism is used to guarantee the correctness of simulation. However, the rollback needs large memory to store undetermined states and makes the debugging of network simulation much more difficult. To save the memory usage and shorten the developing time, we apply the conservative mechanism in HiNetSim.

2.2 Load Balancing

In the parallel simulation, if simulating workloads assigned to different threads/processes are not balance enough, the parallelism efficiency will be greatly decreased. To achieve the load balancing, we deploy both static and dynamic mechanisms.

At the initialization phase of the simulation, a static load balancing mechanism is used. According to different network topologies, HiNetSim provides an effective graph allocation strategy to put *LE*s into the same process based on their communication affinity. The problem statement of the *LE* allocation is: given a graph G with n weighted vertices and m weighted edges, how to divide the vertices into p sets so that every set has similar sum value of vertex weights and sum value of edge weights respectively. This problem is known to be NP complete, but there are some heuristic approximate solutions. We use Chaco (a graph partitioning package) to solve this problem which works well in partitioning hierarchical network topologies.

At the simulating phase, to achieve sub-millisecond granularity workload migration, a cooperated migration mechanism is proposed, which combines the merit of workload sharing and workload stealing. For each thread, our mechanism separates *run_list* and *mig_list* for normal execution and migration. Inside a thread, *LE*s are scheduled and stolen as follows:

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

2.3 Cycle-level Accuracy

To guarantee the cycle-level accurate, simulation in flit level may be the best solution. However, the flit-level simulation introduces plenty of simulation tasks and eventually leads to poor efficiency. To carry out the large-scale simulation with the same accuracy within a reasonable time, we use packet-level simulation in virtual cut through switching (VCT). In addition, VCT is the most commonly used switching technology. Our process of the packet-level simulation is:

1) When the first flit of a packet arrives at one switch, the switch accepts it only if there is enough space for the whole packet in its receiving queue.
2) The switch selects one packet from its receiving queues based on certain arbitration algorithm and calculates the flow control credit to check whether the receiving side has enough space to accept it.
3) If the flow control credit is enough, then the switch sends out the packet flit by flit until the last one is sent out.

In simulation, we assume that all switches in an interconnection network have the same bandwidth. If we define the length of flit as L, the bandwidth of switch as BW, $T_R\ (w, k)$ and $T_S\ (w, k)$ as the time stamps of switch w receiving and transmitting the kth flit respectively, T_C as the time of arbitration delay, routing delay, flow control and etc., then the $T_S\ (w, k)$ can be described as:

$$\begin{cases} T_S(w,0) = T_R(w,0) + T_c & k = 0 \\ T_S(w,k) = T_S(w, k\text{-}1) + L/BW & 1 \le k \le n \end{cases} \quad (1)$$

As shown in the equation (1), $T_S\ (w, n)$ can be determined when $T_R\ (w, 0)$ and T_c is known. If we assume that all the ports of switches in an interconnection network have the same bandwidth, then $T_S\ (w, n) = T_R\ (w, 0) + T_c + (n\text{-}1) \times L/BW$. Therefore, instead of simulating all the flits, the performance result can be obtained by simulating only the first flit and the last flit of each packet. Such simplification can accelerate the simulation without any loss of accuracy.

2.4 Flexibility

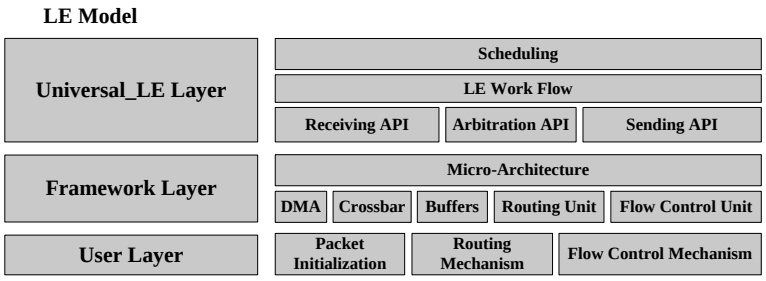


Fig.1. *LE* hierarchical architecture

Network topology, routing algorithm, and flow control mechanism are tightly coupled with each other. In addition, many configurable parameters in network devices are only working with certain topologies and routing algorithms. Thus, achieving the flexibility of supporting various kinds of network architecture is rather difficult.

We implement a hierarchical structure to separate the simulation of microarchitecture of the network device and mechanisms for full simulated network (flow control and routing algorithm) from each other. Hierarchical structure is widely used in computer filed, such as operating systems, network, and etc. In HiNetSim, we borrow this idea and implement the *LE* with three layers. As shown in Figure 1:

• *Universal_LE Model*: takes charge of the scheduling of LE and defines the work flow of simulated network device including sending, receiving, arbitration and etc. The model does not implement any parameters or mechanisms of the target network.

• *Framework Model*: simulates the device micro-architecture including DMA, crossbar, buffers, routing unit and flow control unit and etc.. It also performs the function of performance statistics.

• *User Model*: implements the mechanisms of packet initialization, flow control and routing algorithm.

3 Implementation

3.1 HiNetSim Architecture

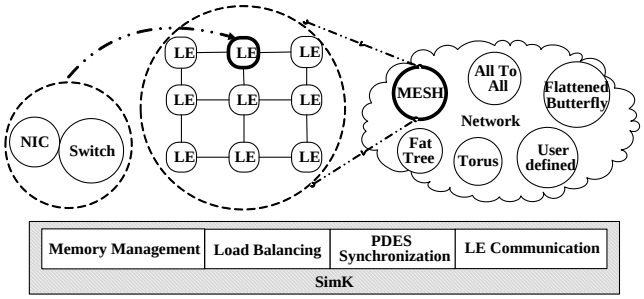


Fig.2. HiNetSim architecture

As is shown in Figure 2, in HiNetSim, *LE*s (can be switch or NIC) are connected with each other following the target network topology and SimK as the simulation kernel schedules *LE*s into execution and performs the synchronization and communication between *LE*s. Regarding a new network architecture, users can build the simulation from following four aspects: topology, routing algorithm, flow control mechanism, and switch/NIC micro-architecture. HiNetSim has implemented several well-known network architectures including fully connected, n-dimension torus, flattened butterfly, fat tree and etc. So, users can easily build their hierarchical networks by reusing codes of the implemented networks in proper network hierarchies.

3.2 Topology Generation

There are two independent topologies establishing modules: basic module and customized module. Basic module implements the configuration of our implemented topologies, including fully connected, torus, fat tree, and etc., while the customized module is used to configure user defined hierarchical topologies. To construct a topology, user can either write codes with HiNetSim's APIs (for complicated regular topologies) or describe the topology in configuration files (for irregular topologies).

3.3 Routing Mechanism

Since the routing algorithm is always tightly combined with topology, four routing mechanisms are supported to achieve the best flexibility:

1) *Source address routing*: fill the destination port vector (record the destination port number in each hop) in the packet structure.
2) *Table-based routing*: fill the address-to-port mapping table in each switch /router structure.
3) *Dimension order routing*: especially for *n*-dimension mesh and torus topologies, fill the packet structure with the description of dimension ordering and fill the dimension-to-port mapping table in each switch/router structure;
4) *Zone routing* [3]: an adaptive routing mechanism for direct networks, fill the zone identification, destination address and guidance bits of dimension ordering in the packet structure and fill the zone masks in each switch/router structure.

3.4 Flow Control Mechanism

HiNetSim implements the credit-based flow control mechanism as the default option. In PDES, two *LE*s are usually running at different timestamps, so we must buffer the flow control packet in a dedicated queue in the destination *LE* and pop it out when the destination *LE* has reached the timestamp of the flow control packet. To simplify the implementation without violating PDES synchronization policy, each flow control packet contains two different timestamps: one indicates the time to use the credit and the other one is the timestamp of the source *LE* for passing the flow control packet to the destination *LE*.

3.5 Simulation of Network Devices

HiNetSim simulates the detailed micro-architectures of NIC and switch. The router in HPC interconnection network is built from several NICs and a switch. As shown in Fig. 3, the NIC model contains seven modules: trace generator (generating artificial traffic patterns, such as uniform random, tornado, bit reversal and etc.), North Bridge (simulation of the I/O bus), DMA model (simulation of a RDMA engine), route generator (generating routing information to be carried with packets), receiver (receiving packets and putting them in the receiving buffers), transmitter (transmitting packet out) and flow control.

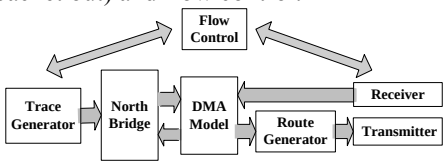


Fig.3. NIC model

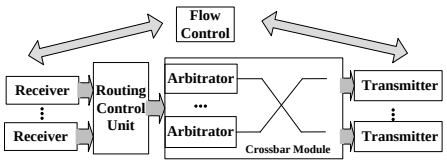


Fig.4. Switch model

Fig. 4 shows an input-queuing switch model in HiNetSim. It contains five modules: receiver (receiving packets and putting them into corresponding virtual

channels), transmitter (performing the output arbitration and transmitting packet out), crossbar (for each transmitter, selecting proper packets to send out), routing control (calculating destination port based on current routing mechanism), and flow control. Parameters such as the number and depth of virtual channels, receiving delay, arbitration strategy (round-robin, matrix arbiter or priority) and delay, internal bus bandwidth and etc. are configurable.

4 Simulation Examples

In this section, we perform simulations on both basic networks (flattened butterfly and 4D torus) and a proposed large-scale hierarchical network (4.5D Torus shown in Section IV.C) to demonstrate the simulation capability of HiNetSim.

4.1 Flattened Butterfly (FB): 1024 Nodes

Table 1. 1024 nodes fattened butterfly

Name	Configurations	# of nodes per switch
FB1	8×8	16
FB2	16×16	4
FB3	8×8×8	2
FB4	4×4×4×4	4

Fig. 5 shows the topology of the two dimensions flattened butterfly [19]. In each row and column, switches are fully interconnected with each other. Table 1 shows 4 different kinds of configurations of 1024 nodes flattened butterfly, including 2 to 4 dimensions. First column in Table 1 shows the net name we called in our simulation. Configuration column shows the number of nodes per dimension, and the last column shows the number of nodes per switch directly connected with. We use VOQ (Virtual Output Queuing) microarchitecture and uniform random traffic [20-22].

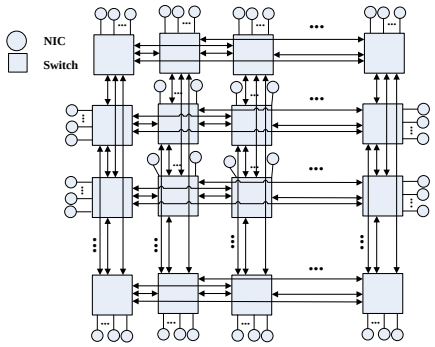


Fig.5. Flattened butterfly topology

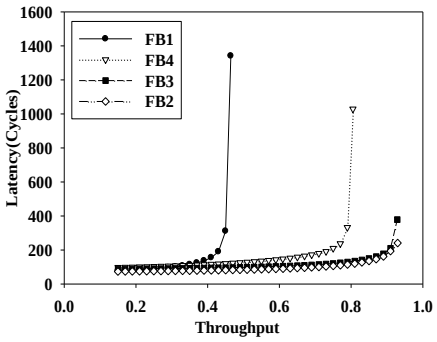


Fig.6. Throughput vs. Latency: FB

Fig. 6 shows the performance of flattened butterfly under different configurations. Network FB1 achieve much lower throughput (0.46) than others, because it is a flattened butterfly with oversubscription that connects 16 nodes per switch (standard fattened butterfly 8×8: 8 nodes per switch). On the contrary, network FB2 and FB3 can achieve throughputs up to (0.91~0.92) with very low latency, because these they are connecting with less nodes than the standard flattened butterfly (standard flattened butterfly 8×8×8: 8 nodes per switch, 16×16: 16 nodes per switch). Network FB4 is a standard flattened butterfly with high link occupation can achieve the throughput of 0.81. Users can perform these simulations by simply configuring the number of dimension and the number of nodes per dimension.

4.2 4D Torus: 4096 Nodes

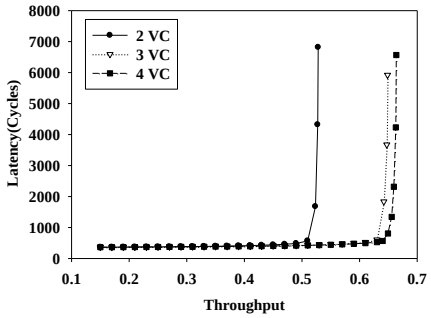


Fig.7. Throughputs vs. Latency: 4D torus with different number of VCs

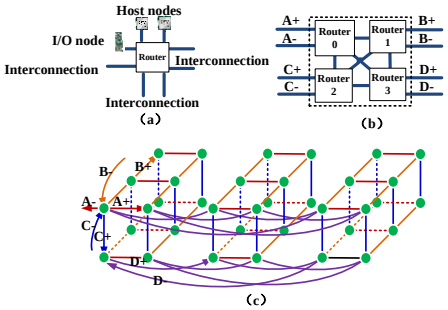
We simulate the 4D Torus network at the 4096 nodes scale (8 × 8 × 8 × 8: each dimension contains 8 nodes) and study the effect of virtual channels on network performance. One VC is used as the escape channel in these simulations. In addition, dimension order routing (DOR) and uniform random traffic is used.

As shown in Fig. 7, with 2 VCs, the network throughput is 0.51. With 3 and 4 VCs, throughputs can increase to 0.63 and 0.65 respectively. The improvement is achieved by reducing the head-of-line (HOL) blocking. However, because of the limitation of using DOR algorithm, the throughput is difficult to reach 0.7.

4.3 4.5D Torus Network: A Hierarchical Network

We proposed a 4.5D Torus network to show the simulation of hierarchical network. The 4.5D Torus topology aims to optimize local communication and build large-scale network with low radix routers. As is shown in Fig. 8 (a), the basic building block of network is the 8-port router (2 ports for the host, 1 port for I/O, and 5 for interconnection). As shown in Fig. 8 (b), 4 fully connected routers form a Superblock (8 nodes). The remaining 8 ports of the Superblock are used for external interconnection. Fig. 8 (c) shows the standard 4D torus topology built with

Superblocks. Since the interconnection inside Superblock is only for local communication, we treat it as 0.5D dimension. The network is of full-system 4D torus plus local 0.5D, so we name it 4.5D Torus.



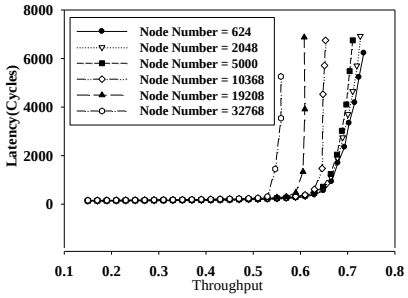


Fig.8. Topology of 4.5D Torus

Fig.9. Throughput vs. Latency: 4.5D torus

This simulation uses zone routing mechanism and 4 virtual channels (including 2 escape VCs). We set the virtual channel's buffer to 4096Byte and MTU (Maximum transmission Unit) to 1024Byte. We perform the simulation with a traffic that contains 70% intra-superblock communication, 20% neighbor-superblock communication within 3 hop counts, and 10% to the remaining superblocks.

Table 2. Configurations of 4.5D torus

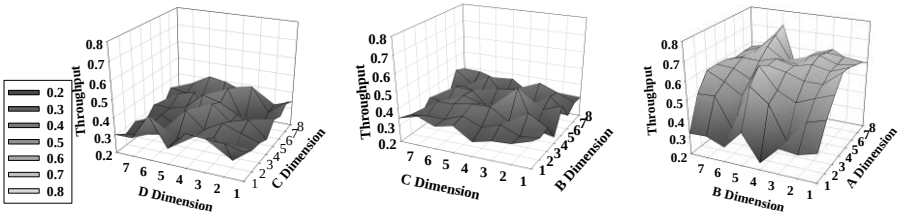
A×B×C×D	# of Superblocks	# of Nodes
3×3×3×3	81	648
4×4×4×4	256	2,048
5×5×5×5	625	5,000
6×6×6×6	1,296	10,368
7×7×7×7	2,401	19,208
8×8×8×8	4,096	32,768

We simulate the 4.5D torus with different scales (648~32,768 nodes). As shown in Table 2, the first column shows the dimension length of *A*, *B*, *C*, and *D*. The other two columns show the number of superblocks and nodes (8 nodes per superblock).

As shown in Fig. 9, when network scale is small (e.g. 648 nodes), the throughput can reach to 0.75. As the network scale increases, the throughput keeps decreasing. When the number of nodes is 32,768, the throughput is only 0.56. The throughput decrement is mainly caused by the network load imbalance introduced by DOR routing algorithm. The ability of gathering network load imbalance information is very important for a simulator. To get heavily loaded devices or paths, HiNetSim provides detailed performance statistics for each network device and even each port. By comparing the throughput of each node, the bottleneck of network can be easily found out. We use HiNetSim to analyze the network load of 4.5D torus with 32,768 nodes. 3D graphic can compare only the loads of two dimensions, so we show the load balancing with three combinations of dimensions: *A* and *B*, *B and C*, *C and D*.

From Fig.10 (a) and (b), we can see that throughputs of *B, C* and *D* dimensions have very small variations. Fig.10 (c) shows that *A* dimension is load imbalanced as

the throughput varies from 0.2 to 0.7. Some nodes are heavily loaded (exceed the saturation point of a router with 2VCs) in *A* dimension and become the bottleneck. Such imbalance is mainly caused by the DOR algorithm (*A* dimension first).



(a)*B* and *C* dimensions **(b)***C* and *D* dimensions **(c)***A* and *B* dimensions

Fig.10. Load balancing condition of 4.5D torus network

5 Performance Evaluations on HiNetSim

This section evaluates the performance of HiNetSim on a 12-core Intel machine. The configuration of system is listed in Table 3.

Table 3. Experiment environment

CPU type	Xeon X5675
CPU number	2
Memory	96GB
OS	CentOS 6.3
Compiler	gcc 4.46
Library	Pthread

Table 4. Network types and configurations

Network Types	**# of Switch numbers**	**# of Nodes**
4.5D Torus	576 (3×3×4×4×4)	1152
Flattened Butterfly	64 (8×8)	1024
Fat Tree	320 (*m*=16,*n*=3)	1024
4D Torus	1296 (6×6×6×6)	1296

The performance of HiNetSim is evaluated with simulations of different types of networks. Table IV shows the network type and configuration of each network. These networks are in the similar scale (1024~1296 node). All these simulations are using 1024Byte packets and 100% inject rate. The parallel speedup is calculated as: execution time on single core / execution time on multiple cores.

As shown in Fig. 11, HiNetSim can achieve linear speedup as the number of processor cores increases. Even super-linear speedup 15.6 is obtained when simulating 1,024 nodes flattened butterfly. The speedup is mainly related with the number of *LE*s: the fewer the *LE*s, the less communication and cache miss. So, 1024 nodes flattened butterfly with the fewest *LE*s can achieve super-linear speedup.

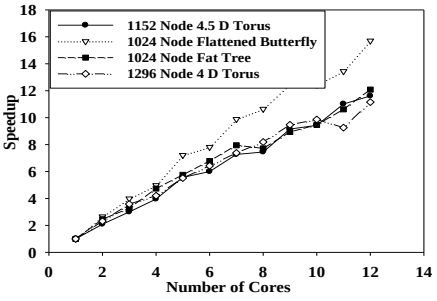


Fig.11. Speedup of various networks

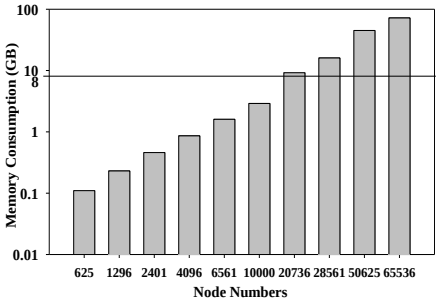


Fig.12. Memory consumption of HiNetSim

Regarding the large scale network simulation, memory consumption is also an important issue. Fig. 12 shows the memory consumption of simulating 4D Torus which involves lots of *LE*s. When simulating no more than 10,000 nodes, less than 8GB memory is used. 8GB memory is easy to be fulfilled even in a personal laptop. When simulating 65,536 nodes, 72GB memory is used, which is also easy to get for a dual-processor blade server. So, with good parallel speedup and acceptable memory consumption, HiNetSim is capable of simulating large interconnection networks.

6 Conclusions and Future Work

HiNetSim is a parallel simulator that simulates large scale hierarchical interconnection networks with high efficiency, accuracy and flexibility. We use packet level simulation to guarantee the simulation efficiency, but still achieve the same accuracy as flit level. To shorten the development time of the simulation on new hierarchical network architectures, we provide flexible topology configuration, general purpose routing algorithm interfaces, and simulations of many commonly used networks. In this paper, we demonstrate the function of HiNetSim by simulating flattened butterfly, 4D Torus and a hierarchical network 4.5D Torus. Evaluation shows that HiNetSim can achieve linear parallel speedup. In addition, it can perform the simulation of 10,000 nodes network with less than 8GB memory and 65,536 nodes network with 72GB memory.

References

1. Top500 list. [Online]. Available: http://www.top500.org
2. D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The IBM Blue Gene/Q interconnection fabric," IEEE Micro, vol. 32, no. 1, pp. 32–43, Jan./Feb. 2012.
3. D. Chen, N. A. Eisley, P. Heidelberger, R. M. Senger, Y. Sugawara, S. Kumar, V. Salapura, D. L. Satterfield, B. Steinmacher-Burow, and J. J. Parker, "The IBM Blue

Gene/Q interconnection network and message unit," in Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal., 2011, pp. 1–10.
4. Y. Ajima, Y. Takagi, T. Inoue, S. Hiramoto and T. Shimizu, "The Tofu Interconnect", IEEE Micro, vol. 32, no. 1, pp. 21-31, Jan./Feb 2012.
5. J. Kim, W. J. Dally, S. Scott, and D. Abts, "Cost-Efficient Dragonfly Topology for Large-Scale Systems," IEEE Micro, vol. 29, no. 1, Jan.2009.
6. BookSim 2.0. [Online]. Available: http://nocs.stanford.edu/booksim.html
7. A. Nayebi, H. Sarbazi-Azad, A. Shamaei and S. Meraji. 2007. "XMulator: An Object Oriented XML-Based Simulator." In Proceedings of the First Asia Interna-tional Conference on Modelling & Simulation. Phuket, Thailand, 128-132.
8. D. Tutsch, D. Ludtke, A. Walter, and M. Kuhm. 2005. "CINSim: A Component-based Interconnection net-work Simulator for Modeling Dynamic Reconfiguration." In ESM'05: European conference on modeling and simulation. Riga, Latvia, 32-39.
9. D. Tutsch, M. Brenner. 2003. "MINSimulate – A Mul-tistage Interconnection Network Simulator." In ESM'03: European Simulation Multiconference: Foundations for Successful Modeling & Simulation. Notingham, SCS, 211-216.
10. F. J. Ridruejo and J. Miguel-Alonso. 2005. "INSEE: An interconnection network simulation and evaluation environment." In Lecture Notes in Computer Science, vol (3648):1014-1023.
11. H. MIWA, R. SUSUKITA, H. SHIBAMURA, et al. "NSIM: An Interconnection Network Simulator for Extreme-Scale Parallel Computers," IEICE Transactions on Information and Systems, v E94-D, n 12, p 2298-2308, December 2011.
12. Fernando Pardo and Jose A. Boluda. SimuRed: A flit-level event-driven simulator for multicomputer network performance evaluation, Computers & Electrical Engineering, Volume 35, Issue 5, September 2009, Pages 803-814, Fernando Pardo, Jose A. Boluda
13. P. Abad, et al. "Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers", In Intl Symposium on Networks-on-Chip, NOCS '12, 2012.
14. Bignetsim [oneline] Available: http://charm.cs.illinois.edu/research/bignetsim
15. ns-3 [on line] Available: http://www.nsnam.org/
16. Nongda Hu, Binzhang Fu, Xiufeng Sui, Long Li, Tao Li, and Lixin Zhang. "DCNSim: a unified and cross-layer computer architecture simulation framework for data center network research." In Proceedings of the ACM International Conference on Computing Frontiers (CF '13). Article 19, 9 pages, 2013.
17. J. Xu, M. Chen, G. Zheng, Z. Cao, H. Lv, and N. Sun, "SimK: a parallel simulation engine towards shared-memory multiprocessor," *Journal of Computer Science and Technology*, vol. 24, no. 6, pp. 1048–1060, 2009.
18. Richard M. Fujimoto, Parallel discrete event simulation, Communications of the ACM, v.33 n.10, p.30-53, Oct. 1990.
19. Kim, John, William J. Dally, and Dennis Abts. "Flattened butterfly: a cost-efficient topology for high-radix networks." ACM SIGARCH Computer Architecture News. Vol. 35. No. 2. ACM, 2007.
20. W. J. Dally and B. Towles, Principles and Practices of Interconnection Networks. San Francisco, CA: Morgan Kaufmann, 2004.
21. W. J. Dally, "Virtual-channel flow control," International Symposium on Computer Architecture, pp. 60-68, 1990.
22. J. Duato, "Deadlock-free adaptive routing algorithms for multicomputers: evaluation of a new algorithm," Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing, pp. 840-847, 1991.