Postprint of Parallel Simulation of Large-Scale Many-Core Architectures

Authors: Ye Xiaochun, Fan Dongrui, Chen Mingyu, Lü Huiwei

Date: 2017-03-10T00:00:00+00:00

Abstract

As the number of processor cores inside chips increases, multi-core processors are gradually evolving toward many-core architectures. This novel many-core architecture presents challenges for computer simulation. Serial simulation can no longer satisfy speed requirements; it is imperative to fully leverage the multi-core resources of existing parallel host machines to enhance simulation speed without compromising simulation accuracy. This paper takes two architectures —many-core and many-core clusters—as examples to demonstrate the necessity and feasibility of parallel simulation technology in the simulation of computer parallel architectures. In many-core simulation, we achieve unchanged accuracy while increasing simulation speed by 10×; in many-core cluster simulation, the total number of simulated small processor cores reaches the thousand-core scale, and a hybrid programming and execution environment is implemented, providing a foundation for scalability testing of this architecture.

Full Text

Preamble

Vol. 9 No. 6
Information Technology Letters

Parallel Simulation of Large-Scale Many-Core Architectures
Ye Xiaochun, Fan Dongrui, Chen Mingyu, Lü Huiwei

Abstract

As the number of processor cores on a chip continues to increase, multi-core processors are evolving toward many-core architectures. This new architectural paradigm presents significant challenges for computer simulation. Serial simulation can no longer meet the demands for speed, making it imperative to

leverage the multi-core resources of existing parallel host machines to accelerate simulation without sacrificing accuracy. This paper uses both many-core and many-core cluster architectures as examples to demonstrate the necessity and feasibility of parallel simulation techniques in computer architecture simulation. For many-core simulation, we achieve a 10× speedup while maintaining unchanged accuracy. For many-core cluster simulation, we simulate a total of one thousand processor cores and implement a hybrid programming and execution environment, providing a foundation for scalability testing of this architecture.

Keywords: many-core, many-core cluster, parallel simulation

1 Introduction

Simulators play a pivotal role in processor architecture research. Simulation technology permeates the entire system development process: during early development, simulators are used for coarse-grained modeling to select optimal solutions; during development, they verify various microarchitectural designs; in later stages, they support software development and testing. After the hardware system becomes operational, simulators can obtain profiling information that is inaccessible through hardware alone, enabling bottleneck analysis and performance optimization. In fundamental computer science research, simulators can model both existing and future novel architectures, thereby advancing microarchitectural and system software studies. With their controllable and reproducible characteristics, simulators serve as essential tools for gaining insight into system behavior.

The performance of simulators is primarily measured by two metrics: simulation speed and simulation accuracy. In traditional serial simulation, these two metrics are often mutually exclusive—improving speed typically leads to accuracy loss, and vice versa. Therefore, developing a simulator that guarantees both speed and accuracy represents a significant research challenge. Given today' s abundant parallel computing resources, parallel simulation emerges as a viable solution. After introducing a parallel framework, a further question is how to simulate even larger-scale many-core architectures to achieve the goal of thousand-core simulation, which constitutes another important focus of this paper.

2.1 Graphite

The Graphite simulator is an open-source simulator released by MIT in 2010 [1], targeting performance in the range of 100 to···

2.1.1 Graphite Execution Model

Graphite is a fast, high-level, large-scale multi-core simulator. From the perspective of applications running on the simulator, they are completely unaware of the host machine they are executing on. The host can be either a multi-core

machine that parallel-simulates a large-scale many-core architecture SMP, or a cluster, without requiring any modifications to the application. This is similar to the SimK parallel simulation framework [2]. In SimK, running the same test program on a single host versus a cluster host requires no modifications to the simulated program—only changes to the SimK API calls are needed. In Graphite, each application thread is mapped to a simulated processor core, with one or more threads mapped to a process. Multiple processes can run on different hosts or on the same host, making Graphite a two-level parallel simulation architecture.

Additionally, Graphite uses a dynamic binary translator to generate instructions and optimizes application data access. It statically partitions the application address space across different hosts and places data associated with that address space on the corresponding hosts, caching frequently accessed data in local memory. This addresses potential bottlenecks caused by frequent access to data residing on different hosts.

2.1.2 Graphite Synchronization

[Figure 1: see original paper] Graphite provides three different synchronization models, allowing users to select the most suitable one for their needs.

Lax Synchronization: This synchronization approach only synchronizes local clocks when synchronization events occur in the application (including locks, barriers, message reception through message-passing APIs, thread creation and termination). Consequently, it has minimal synchronization frequency and overhead, resulting in the smallest simulation slowdown. However, these events can occur out of order, leading to reduced simulation accuracy.

LaxP2P: This is a distinctive synchronization method. Observation reveals that timing inaccuracies are primarily caused by a small number of threads. Therefore, every few clock cycles, each target core randomly pairs with another core for comparison. If their cycle counts differ significantly, the core with the larger cycle count pauses execution for a period. This approach achieves good performance while maintaining reasonable simulation accuracy.

LaxBar: This is a barrier-like synchronization method. Every few cycles, all target cores synchronize once. This ensures sufficient synchronization among cores, and when the synchronization interval equals 1, the entire simulator can achieve cycle-accurate simulation precision. However, this frequent synchronization directly leads to performance degradation.

2.2 COTson

COTson is a full-system simulation architecture developed by HP based on AMD' s SimNow simulator [3]. It can simulate single-core, multi-core, and even clusters with interconnect networks. The simulator features a "pluggable" architecture, meaning users can replace existing modules with their own designs.

A key design principle of COTson is the trade-off between speed and accuracy—that is, sacrificing simulation accuracy for speed or vice versa. This is a popular design principle today. With the rise of many-core architectures and increasing simulation scales, completing large-scale simulations within tolerable timeframes has become a research hotspot, making such trade-offs essential.

2.2.1 COTson Simulation Techniques

COTson uses the SimNow virtual machine for functional simulation and its own timing backend to determine the performance of simulation targets. Aiming for large-scale full-system simulation, COTson employs sampling—a technique frequently used in fast simulation—to improve speed. The sampling method couples the sampling mechanism with the simulator, collecting instruction and memory access information from the running simulator and passing this information to the timing backend to rapidly obtain timing data.

2.2.2 COTson Synchronization

In many-core simulation, the most commonly adopted method is the Parallel Discrete Event Simulation (PDES) algorithm [4]. However, accurate PDES algorithms incur significant overhead during synchronization, slowing down simulator execution. This becomes a serious problem in large-scale simulations. Therefore, some simulators that do not prioritize timing accuracy improve upon this algorithm, sacrificing some precision for faster simulation speed.

The synchronization method used in COTson dynamically adjusts synchronization granularity. As the name suggests, "dynamic"means using larger granularity synchronization for simulation parts of less interest, while using smaller granularity synchronization for parts of greater interest, thereby achieving faster simulation speed.

From these two important simulation techniques—sampling and synchronization —it can be said that COTson's primary philosophy is to identify simulation points of interest and relax accuracy requirements for unimportant parts, achieving a balance between precision and speed.

2.3 Other Simulators

BGLsim [5] is a BlueGene/L simulator developed by IBM that runs on real Linux cluster systems, with multi-node communication implemented using MPI. BGLsim simulates all hardware of the large-scale cluster system BlueGene/L, runs the BlueGene/L Linux system on this virtual hardware, and provides a ported BlueGene/L MPI library.

MPI-SIM [6] is an MPI library developed by UCLA, primarily used for testing, debugging, and predicting the performance of parallel programs on various architectures. It can provide different simulation results based on target

system parameters, with main parameters including processor count and communication latency. MPI-SIM is a parallel simulator that also provides a new conservative synchronization algorithm to reduce synchronization overhead and frequency.

Simics [7], originally developed by the Swedish Institute of Computer Science, is a full-system simulator that can simulate numerous platforms and run various operating systems without modification. Simics can simulate multiple instruction set architectures and has rich peripheral support. For parallelism, Simics also provides a Link mechanism-based parallel simulation method.

3 Many-Core Simulation

The simulators mentioned above each have their own characteristics, and we have also conducted our own experiments in many-core simulation. We have accumulated experience in parallel frameworks and many-core simulation, developing the serial many-core Godson-T simulator (GAS) [8,9] and the SimK parallel simulation framework [2]. To further accelerate many-core simulation, we directly parallelized the GAS simulator using SimK, resulting in a parallel many-core simulator that balances both simulation speed and accuracy: P-GAS.

3.1 SimK Parallel Simulation Framework

SimK is an open-source parallel simulation framework designed to support efficient parallel simulation [2] while maximizing generality and usability. It provides support for ultra-large-scale simulation, high scalability, heterogeneous host support, and generic, concise interfaces.

SimK employs a Parallel Discrete Event Simulation (PDES) synchronization mechanism. The core idea of this algorithm is that if all local parts of a system are synchronized, then the entire system is synchronized. SimK uses a conservative PDES synchronization scheme, where local causality must be strictly guaranteed. Any operation that might violate causality is prohibited, and a logical unit is blocked from execution until it is safe to process the event with the smallest timestamp in its event list.

Additionally, to make the simulation framework more efficient, SimK highly optimizes Pthread using lock-free synchronization mechanisms and supports user-level thread scheduling.

SimK provides an Application Programming Interface (API) that facilitates the parallelization of modular simulators. This section primarily introduces how to use the SimK simulation framework to parallelize the Godson-T many-core simulator GAS.

3.2 GAS Simulator

The Godson-T simulator (GAS) is an event-driven simulator that serially simulates the Godson-T chip. The Godson-T chip is a high-performance many-core chip containing 64 processor cores with a 2D-Mesh on-chip network structure. Communication between processor cores and with L2 cache occurs through on-chip routers, as shown in [Figure 2: see original paper]. The goal of the GAS simulator is to provide a modular, configurable simulation tool for Godson-T development. It uses an event-driven approach to simulate the target system, where simulation models transition from one state to another driven by events. In many-core systems, events are generated in different simulation components, including processor cores, routers, and L2 caches. GAS uses a global queue to control event processing and is timing-accurate.

The P-GAS simulator uses the SimK simulation framework to parallelize the Godson-T many-core simulator GAS. The parallelization objectives are: to partition the modules in the Godson-T simulator across different threads, run simulation tasks in parallel, achieve good speedup, and ensure correct results without compromising simulation accuracy. During parallelization, to achieve better scalability, modifications to both the original simulator code and the parallel simulation framework code should be minimized.

3.3 P-GAS Parallelization Process

Module Partitioning: To improve the efficiency of the P-GAS simulator, factors such as component correlation, communication volume, and load balance must be considered during module partitioning. Improper handling of any factor may lead to load imbalance, high communication volume, or high synchronization overhead in the parallelized system.

After module partitioning, each subsystem needs to be equipped with an independent event queue and integrated with SimK synchronization communication APIs to be encapsulated as a SimK-style module, as shown in [Figure 3: see original paper].

All inter-module communication in SimK occurs through SimK communication channels. When a module generates a message, it places the message into the buffer of the corresponding channel. When another module connected to the channel is scheduled, it checks all its communication channels and retrieves all received messages.

In the serial version of GAS, all modules share a single global queue. As the queue becomes long during simulator execution, event insertion overhead becomes excessive. After module partitioning in the P-GAS simulator, a local queue is added for each module, eliminating the original global queue and managing queues on a per-module basis. Since the load of most modules is relatively balanced, queue length and memory usage can be conveniently configured and managed.

Improving SimK' s Synchronization Communication Mechanism for GAS Specifics: The P-GAS simulator continues to use SimK' s conservative lookahead synchronization. The prerequisite for this mechanism to work correctly is that a module will not receive events with timestamps smaller than its current local time. The GAS simulator contains zero-delay events, primarily used to simulate signal lines sending REQ/ACK or REQ/NACK signals. These events cause the module sending REQ to receive ACK/NACK from other modules after its clock has advanced, at which point the module time is greater than the timestamp of the event to be processed, directly causing SimK's conservative lookahead synchronization mechanism to produce runtime errors in the parallel GAS simulator.

The solution introduces a zero-delay event counter. Each module is augmented with a flag variable initialized to 0. When the module' s processing function needs to send REQ requests to other modules, the flag is incremented by 1. Each time an ACK or NACK is received, the flag is decremented by 1. Based on this flag, the module can easily determine whether events requested from other modules will still arrive in the current cycle, thereby deciding whether the clock can advance.

4 Many-Core Cluster Simulation

After applying parallel discrete event simulation algorithms to accelerate single-chip many-core simulators, we sought to increase the number of processor cores to verify the scalability of this method as chip scale expands and to conduct exploratory research on thousand-core architectures. For existing simulators, we set thousand-core simulation as a target. Intuitively, two solutions exist for studying thousand-core architectures: first, extend the original GAS simulator to a serial simulator with 1024 processor cores, then parallelize it using SimK to complete single-chip thousand-core simulation; second, interconnect multiple GAS simulators in a cluster configuration to form a many-core cluster simulator, which can also be accelerated using SimK as the parallel framework.

After comparative analysis, we adopted the second solution. The first approach requires extensive modifications to the existing serial simulator and offers poor scalability. In contrast, cluster-based simulation only requires adding some modules to the existing serial simulator and enables the simulator to work conveniently across multiple hosts, ensuring good scalability.

Our goal is to develop a cycle-accurate many-core cluster simulator based on the GAS simulator and SimK to simulate Godson-T clusters and port communication libraries to run simple applications.

The target system architecture we aim to simulate is shown in [Figure 4: see original paper]. To this end, we developed a many-core cluster simulator called P-Gcluster.

4.1 P-Gcluster NIC and Switch Module Design

The original GAS simulator lacks external interaction modules. Therefore, to compose a thousand-core-scale cluster by interconnecting multiple GAS simulators, we must add a NIC module to the GAS simulator, as shown in [Figure 5: see original paper]. This module primarily simulates the NIC' s control registers, status registers, and data buffer.

Connecting the NICs of various GAS simulators via switches forms a many-core cluster. The NIC serves as a bridge for data transmission between the GAS simulator and the switch. In the GAS simulator, the process of sending a data packet via NIC can be summarized in three steps: first, the GAS simulator writes data to the NIC buffer; second, the GAS simulator sends an event to the NIC; third, the NIC sends the data packet to the switch. The detailed process is shown in [Figure 6: see original paper].

Different simulation accuracies are reflected in the packet transmission step from the NIC, but the NIC module is identical to the processor core. In NIC design, we developed two types based on different simulation granularities: simple NIC and basic NIC. Both use mathematical modeling to simulate delays at each stage, while the basic NIC adds packet queuing delay in the buffer and flow control mechanisms compared to the simple NIC.

To accurately simulate the NIC' s operation in real hardware, we incorporated flow control mechanisms into our basic NIC. Flow control is a synchronization protocol between network components for sending and receiving flow control units, determining when data packets can be transmitted. Its purpose is to ensure successful data transfer from sender to receiver without causing buffer overflow—that is, flow control is a way to exchange buffer status information between transceivers.

In our system, we adopted a credit-based flow control mechanism [10]. Credit is initially set to the number of receive buffers, representing the receiver' s capacity to accept packets. The working principle of this mechanism is shown in [Figure 7: see original paper].

In the simulator, the NIC module serves as a supplement to the GAS simulator, handling communication between the GAS simulator and the external world. Multiple GAS simulators require a switch module for interconnection. In the system, the switch module functions as a SimK component, registered in SimK and scheduled by SimK.

Switch functionality can be divided into data forwarding and control. The data forwarding function, responsible for fast data transfer, consists of switching fabric and queuing structure. The control function handles switch configuration management and routing table population, generally implemented in software and not discussed here.

Based on the scale of 64 processor cores per GAS simulator, we need 16 GAS sim-

ulators to reach 1024 processor cores in the cluster simulator. Therefore, when designing the switch module, we configured it with 16 ports, each connecting to the NIC module of a GAS simulator.

When SimK schedules switch execution, the switch polls its 16 ports to check for arriving packets. Upon packet arrival, the switch receives the packet, places it into the switch buffer queue, and repeats checking for unprocessed packets until all packets are received. It then identifies the timestamps on received packets. If a timestamp matches the current time, the packet is sent in the current cycle; otherwise, the switch' s current invocation ends until the next SimK scheduling.

As mentioned above, to avoid packet loss during transmission and more accurately simulate real hardware, we added flow control mechanisms to the basic NIC. To cooperate with the flow control mechanism in the basic NIC, the switch module also added flow control support. To simulate packet queuing delay in the switch, we set an upper limit on the number of packets the switch can send within one clock cycle. Within one clock cycle, even if packets remain unprocessed, the switch must stop working for that cycle and increment the timestamp of all unprocessed packets by 1 for continued transmission in the next cycle.

4.2 P-Gcluster Inter-Chip Network Design

When interconnecting multiple GAS simulators, the first step is to implement a non-blocking network using only mathematical analysis of packet transmission delay without simulating packet contention for buffers. Multiple nodes in a many-core cluster can be interconnected in various ways. The simplest and lowest-cost structure is a bus-based interconnect. While transmission delay can remain relatively low under contention-free conditions, bus contention becomes increasingly severe as node count grows, and the bus becomes a bottleneck for inter-node communication, hindering scalability. For cluster systems, several interconnection options are generally available: crossbar interconnect, 2D-Mesh interconnect, or fat-tree interconnect.

Considering the characteristics of these interconnection options and our current simulation scale, we selected the fat-tree interconnect. Fat-trees have been widely adopted in many high-performance computing systems. Their main feature is constant bisection bandwidth, overcoming the shortcomings of conventional binary or multi-way trees where root nodes can become system bottlenecks, redundant paths are lacking, and fault tolerance is poor. Communication bandwidth from leaf nodes to root nodes gradually increases, containing numerous redundant links that provide a foundation for reliability. Moreover, fat-trees can satisfy requirements for high bandwidth, low latency, and scalability.

The overall topology of P-Gcluster is shown in [Figure 4: see original paper]. Each blade consists of 4 GAS simulators, with 16 blades forming a small cluster. Within the cluster, a typical m-port n-trees structure can be employed. In our simulation, the cluster architecture uses a single-level fat-tree with a 16-port 1-tree structure.

4.3 P-Gcluster Two-Level Parallel Environment

With NICs and interconnect networks in place, the cluster simulator platform is essentially complete. However, running and testing programs on this platform requires communication library support for development and execution. Within a single GAS, after SimK parallelization, pthread-based multi-threaded execution is already supported. However, in the cluster simulator, to achieve scalability goals, we ported MPI [11] to the cluster simulator for application development and execution on this platform.

First, to test six basic MPI functions, we ported a simplified version of MPI—gMPI—to the many-core cluster simulation platform. Testing with this simplified MPI verified that MPI could run correctly on the platform. We then ported the widely-adopted MPICH to better support all MPI functions and facilitate application development on the cluster simulator.

Through MPICH porting, we established a two-level hybrid programming environment: inter-chip communication between multiple processor chips uses message passing, while processor cores within a single chip run in parallel via pthread, which aligns well with the many-core cluster architecture.

4.4 P-Gcluster Multi-Process Multi-Host Extension

Due to limited host resources, as the scale of the simulated system grows, the number of nodes to be simulated may exceed the number of physical processor cores available on a single host. When simulating on a single host, host performance becomes a bottleneck. To further improve simulation performance, we need to extend across hosts, simulating the target cluster system on a host cluster.

In cross-host simulation, the following issues must be addressed:

2. When running simulators on multiple hosts, a unified runtime environment is needed to support a single system image, responsible for numbering each host, task allocation, and supporting collective operations in parallel programming.

5 Experiments

The two most important evaluation metrics for simulators are simulation speed and accuracy. We tested both the many-core simulator platform and the many-core simulation cluster described in this paper, with results presented below.

For the many-core simulator P-GAS, we used the SPLASH-2 Kernel [12] parallel program test suite, a widely-used benchmark for testing distributed shared-memory multi-core processors. The test programs include many highly-utilized scientific computing applications, making them well-suited for testing high-performance computing systems. Our experimental platform is a 4-CPU AMD

Opereton 8347 SMP system (16 cores total). [Figure 8: see original paper] shows the speedup achieved when running SPLASH-2 Kernel on the P-GAS simulator.

lists the cycle count differences when running SPLASH-2 on the P-GAS simulator with different thread counts compared to the original serial GAS simulator. As shown, P-GAS achieves an average 10× performance improvement when running with 16 threads while maintaining high accuracy. This demonstrates a parallel many-core simulator that meets requirements in both speed and accuracy.

For many-core cluster simulation, we primarily tested the scalability of P-Gcluster. We tested benchmark cases including cpi, dot product, and matrix multiplication on the many-core cluster simulator, with performance results shown in [Figure 9: see original paper].

Cycle count error of SPLASH-2 running on multi-threaded P-GAS compared to serial version

Benchmark	16-thread error
FFT*	0.07%
Radix sort*	0.01%
LU decomposition	0.17%
Triangular decomposition	0.01%
...	...

Fast Fourier Transform, Radix

[Figure 9: see original paper] shows the speedup of multi-threaded execution on P-Gcluster. The benchmarks include cpi (algorithm for computing) and fox (a parallel matrix multiplication algorithm).

We also conducted tests for multi-process scenarios, primarily testing the scalability of dot product and matrix multiplication under multi-process execution, with results shown in [Figure 10: see original paper]. As the number of processes increases, the amount of data that needs to be transferred between processes in the many-core cluster simulator grows, indicating that further optimization of the NIC and interconnect network remains necessary.

6 Conclusion

Many-core has become the trend in chip development, and simulators play a critical role in chip development. How to improve the simulation efficiency of many-core simulators and expand their simulation scale has become a research hotspot. P-GAS is the parallel version of the Godson-T architecture simulator. To make P-GAS more efficient, we partitioned the global queue of the GAS simulator and performed effective module partitioning based on the GAS simulator

topology. Building upon the efficient SimK parallel framework, we effectively solved the zero-delay synchronization problem in the P-GAS simulator.

We analyzed the parallelization effectiveness from both speedup and accuracy loss perspectives. The simulator achieved the target of average 10× speedup, with maximum speedup reaching 13.6×. In terms of accuracy, the maximum loss is only 0.5%, with average loss not exceeding 0.1%, which does not significantly impact architecture research using the simulator.

After 2010, "how to simulate thousand cores"became a hot topic in the simulator community. Although the GAS simulator targets many-core architectures, its 64-core count still falls short of the thousand-core target. Therefore, building upon the P-GAS simulator work, we developed a many-core cluster simulator, again using SimK as the parallel simulation framework. As the number of simulated processor cores increases, the simulation cluster can be deployed on host clusters, achieving many-core cluster simulation on multi-core clusters and fully leveraging existing multi-core cluster platforms to rapidly and accurately complete thousand-core parallel simulation.

References

[1] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald III, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in HPCA ' 10: The 16th IEEE International Symposium on High-Performance Computer Architecture, 2010.

[2] 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.

[3] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "Cotson: infrastructure for full system simulation," SIGOPS Oper. Syst. Rev., vol. 43, no. 1, pp. 52–61, 2009.

[4] R. M. Fujimoto, "Parallel Discrete Event Simulation" . Commun, ACM, 1990. 33: 30-53.

[5] Ceze, L., et al. Full Circle: Simulating Linux Clusters on Linux Clusters. Proceedings of the Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003, 2003.

[6] Bagrodia, S.P.a.R. MPI-SIM: Using Parallel Simulation to Evaluate MPI Programs. Winter Simulation Conference, 1998: p. 467-474.

[7] Magnusson, P.S.C., M.; Eskilson, J.; Forsgren, D. Simics: A full system simulation platform. Computer IEEE, 2002. vol.35, no.2: p. 50-58.

[8] Dongrui Fan, Nan Yuan, Junchao Zhang, et al. Godson-T: An Efficient Many-Core Architecture for Parallel Program Executions. Journal of Computer Science and Technology (JCST), 2009, vol.24, no.6, pp.1061-1073.

[9] Dongrui Fan, Hao Zhang, Da Wang, Xiaochun Ye, Fenglong Song, Junchao Zhang, and Lingjun Fan. High-Efficient Architecture of Godson-T Many-Core Processor, In Proceedings of 23rd Symposium on Hot Chips, August 2011.

[10] H.T. Kung, T. Blackwell, A. Chapman, "Credit-Based Flow Control for ATM Networks: Credit Update Protocol, Adaptive Credit Allocation, and Statistical Multiplexing". Proceedings of the ACM SIGCOMM 1994 Symposium on Communications Architectures, Protocols, and Applications, Aug. 1994: 101-114.

[11] 都志辉, "高性能计算之并行编程技术——MPI 并行程序设计", 清华大学出版社, 2001.

[12] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36, Santa Margherita Ligure, Italy, June 1995.

Author Biographies:

Ye Xiaochun: Assistant Researcher, Institute of Computing Technology, Chinese Academy of Sciences, yexiaochun@ict.ac.cn

Fan Dongrui: Associate Researcher and Ph.D. Supervisor, Institute of Computing Technology, Chinese Academy of Sciences, Fandr@ict.ac.cn

Chen Mingyu: Researcher and Ph.D. Supervisor, Institute of Computing Technology, Chinese Academy of Sciences, cmy@ict.ac.cn

Lü Huiwei: Ph.D. Student, Institute of Computing Technology, Chinese Academy of Sciences

Note: Figure translations are in progress. See original paper for figures.