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

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

Date: 2016-06-08T00:00:00+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 poses challenges for computer simulation. Serial simulation can no longer meet the speed requirements; it is necessary to fully utilize the multi-core resources of existing parallel host machines to improve simulation speed without compromising simulation accuracy. This paper takes two architectures —many-core and many-core clusters—as examples to illustrate the necessity and feasibility of parallel simulation technology in the simulation of computer parallel architectures; in many-core simulation, maintaining accuracy while increasing simulation speed by 10×; in many-core cluster simulation, the total number of simulated 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
Xiaochun Ye, Dongrui Fan, Mingyu Chen, Huiwei Lü

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, necessitating the full utilization of multi-core resources on existing parallel host machines to improve

simulation speed 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, the total number of simulated processor cores reaches the thousand-core scale, and we 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 microarchitectures; 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 microarchitecture and system software studies. With their advantages of controllability and reproducibility, simulators serve as essential tools for gaining insights 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 simulation speed typically results in accuracy loss, and vice versa. Therefore, developing a simulator that guarantees both speed and accuracy represents a worthwhile research problem. Given today' s abundant parallel computing resources, parallel simulation emerges as a clearly viable solution. After introducing a parallel framework, a further challenge 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 [**?**], with target performance ranging from 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 an SMP (Symmetric Multi-Processing) architecture or a cluster, without requiring any modifications to the application. This is similar to the SimK parallel simulation framework [**?**], where running the same test program on either a single host or a cluster host requires no changes to the simulated program—only modifications 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] illustrates the three-layer architecture of Graphite. 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 via message-passing APIs, thread creation and termination). Consequently, synchronization frequency is low and overhead is minimal, 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 imprecision is 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 [**?**]. 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 —sacrificing simulation precision for speed, or vice versa. This is a popular design principle today. With the rise of many-core architectures and increasing simulation scale, completing large-scale simulations within tolerable timeframes has become a research hotspot, making such trade-offs essential.

2.2.1 Simulation Techniques in COTson

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

2.2.2 Synchronization in COTson

In many-core simulation, the most commonly adopted method is Parallel Discrete Event Simulation (PDES) [**?**]. However, precise PDES algorithms incur significant overhead during synchronization, slowing down simulator execution —a serious problem in large-scale simulation. Therefore, some simulators that do not emphasize timing accuracy improve this algorithm by 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 uninteresting simulation parts and smaller granularity for interesting parts, thereby achieving faster simulation speed.

From these two important simulation techniques—sampling and synchronization —it is clear that COTson' s primary philosophy is identifying simulation points of interest and relaxing accuracy requirements for unimportant parts to achieve both precision and speed.

2.3 Other Simulators

BGLsim [**?**] 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 BlueGene/L cluster system, runs the BlueGene/L Linux system on this virtual hardware, and provides a ported BlueGene/L MPI library.

MPI-SIM [**?**] is an MPI library developed by UCLA primarily for testing, debugging, and predicting parallel program performance across various architec-

tures. It can provide different simulation results based on target system parameters, mainly 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 [**?**], 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 supports multiple instruction set architectures and rich peripheral support. For parallelism, Simics also provides Link mechanism-based parallel simulation methods.

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) [**?**, **?**] and the SimK parallel simulation framework [**?**]. To further accelerate many-core simulation speed, 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 [**?**] while maximizing generality and usability. It provides support for ultra-large-scale simulation, high scalability, heterogeneous host support, and universal, 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, 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 ensured that the event with the smallest timestamp in its event list can be safely processed.

Additionally, to improve simulation framework efficiency, SimK highly optimizes Pthread with 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 Network-on-Chip (NoC) 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.

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

3.3 P-GAS Parallelization Process

Module Partitioning: To improve P-GAS simulator efficiency, module partitioning must consider component correlation, communication volume, and load balancing. 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 requires an independent event queue and SimK synchronization communication API to be encapsulated into a SimK-style module, as shown in [Figure 3: see original paper].

In SimK, all inter-module communication occurs through SimK communication channels. When a module generates a message, it places the message into the corresponding channel' s buffer. When the connected module is scheduled, it checks all its communication channels and retrieves all received messages.

In the serial GAS simulator, all modules share a single global queue, which becomes excessively long during simulation, causing high overhead for event insertion. After module partitioning in P-GAS, we add a local queue for each module and eliminate the original global queue, managing queues on a per-module basis. Since most modules have relatively balanced loads, queue length and memory usage can be conveniently configured and managed.

Improving SimK' s Synchronization Communication Mechanism for GAS Specifics: P-GAS 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. 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 parallel GAS simulation.

The solution introduces a zero-delay event counter. Each module adds a flag variable initialized to 0. When the module' s processing function needs to send REQ requests to other modules, the flag increments by 1. Each time an ACK or NACK is received, the flag decrements 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 aim 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 many-core chips. For existing simulators, we set thousand-core simulation as a target. Intuitively, there are two solutions 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 approach. The first scheme 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 work to simulate Godson-T clusters and port communication libraries to run simple applications. The target system architecture is shown in [Figure 4: see original paper]. To this end, we developed a many-core cluster simulator called P-Gcluster.

4.1 Design of Network Card and Switch Modules in P-Gcluster

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 network card module to GAS, as shown in [Figure 5: see original paper]. This module primarily simulates the network card' s control registers, status registers, and data buffers.

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

Different simulation precision is reflected in the packet sending step of the network card, but the network card module is identical to the processor core. In network card design, we developed two types based on simulation granularity: simple and basic network cards. Both use mathematical modeling to simulate delays at each stage, while the basic network card adds packet queuing delay in the buffer and flow control mechanisms compared to the simple version.

To accurately simulate the hardware network card working process, we added flow control mechanisms to our basic network card. 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 method for exchanging buffer status between transceivers.

Our system adopts a credit-based flow control mechanism [**?**]. Credit is first set to the number of receive buffers, representing the receiver' s capacity to accept packets. The working principle is shown in [Figure 7: see original paper].

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

Switch functionality can be divided into data forwarding and control. Data forwarding enables rapid packet delivery, consisting of switching fabric and queuing structure, while control handles configuration management and routing table population. The control function is generally implemented in software and is not discussed here.

Given that a single GAS simulator contains 64 processor cores, 16 GAS simulators are needed to reach the 1024-core scale in cluster simulation. Therefore, when designing the switch module, we configured it with 16 ports, each connecting to a network card module of a GAS simulator.

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

invocation ends until the next SimK scheduling.

As mentioned earlier, to avoid packet loss during transmission and more accurately simulate real hardware, we added flow control mechanisms to the basic network card. To cooperate with this flow control mechanism, 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 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 Inter-chip Interconnection Network Design in P-Gcluster

When interconnecting multiple GAS simulators, the first step is to implement a non-blocking network that only provides mathematically analyzed packet transmission delay without simulating buffer contention. Many-core cluster nodes can be interconnected in various ways. The simplest and lowest-cost structure is bus-based interconnection, which maintains relatively low transmission delay under contention-free conditions. However, as node count increases, bus contention becomes more pronounced, and the bus becomes a bottleneck for inter-node communication, hindering scalability. For cluster systems, typical interconnection options include crossbar, 2D-Mesh, or fat-tree.

Considering the characteristics of these interconnections and our current simulation scale, we selected the fat-tree approach. Fat-tree has been widely adopted in many high-performance computing systems. Its main feature is constant bisection bandwidth, overcoming drawbacks of binary or multi-way trees where root nodes can become bottlenecks, lack redundant paths, and have poor fault tolerance. Fat-tree gradually increases communication bandwidth from leaf to root nodes, includes numerous redundant links, and provides a foundation for reliability while meeting 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 used. In our simulation, the cluster employs a single-level fat-tree 16-port 1-tree structure.

4.3 Two-Level Parallel Environment on P-Gcluster

With network cards 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 GAS, after SimK parallelization, pthread multi-threading execution is already supported. But in the cluster simulator, to achieve scalability goals, we ported MPI [**?**] to the cluster simulator for application development.

First, to test six basic MPI functions, we ported a simplified MPI version—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 built 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, aligning well with the many-core cluster structure.

4.4 Multi-Process Multi-Host Extension of P-Gcluster

Due to limited host resources, as the scale of the simulated system grows, if the number of nodes to be simulated exceeds the number of physical processor cores on a host, the host performance becomes a bottleneck when simulating on a single host. 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:

1. When running simulators across 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 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 [**?**] 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 Opteron 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 P-GAS with varying 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 for both speed and accuracy.

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

We also conducted tests for multi-process scenarios, primarily examining the scalability of dot product and matrix multiplication under multiple processes, 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 opportunities exist for the network cards and interconnect network.

6 Conclusion

Many-core has become the trend in chip development, and simulators play a critical role in chip development. Improving simulation efficiency and expanding simulation scale have become research hotspots. P-GAS is the parallel version of the Godson-T architecture simulator. To enhance P-GAS simulation efficiency, we split GAS' s global queue and performed effective module partitioning based on GAS topology. Building upon the efficient SimK parallel framework, we effectively solved the zero-delay synchronization problem in P-GAS.

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

After 2010, "how to simulate thousand-core systems" became a hot topic in the simulator community. Although GAS' s simulation target is many-core, 64 cores still fall short of the thousand-core scale. Therefore, based on P-GAS, 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, fully leveraging existing multi-core cluster platforms to rapidly and accurately complete thousand-core parallel simulation.

References

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.

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.

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.

R. M. Fujimoto, "Parallel Discrete Event Simulation" . Commun, ACM, 1990.

33: 30-53.

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.

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

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.

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.

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.

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.

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.

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

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

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

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

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

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