SimICT: A Fast and Flexible Framework for Performance and Power Evaluation of Large-Scale Architecture

Xiaochun Ye, Dongrui Fan, Ninghui Sun, Shibin Tang, Mingzhe Zhang, and Hao Zhang
State Key Laboratory of Computer Architecture, Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China
{yexiaochun, fandr, snh, tangshibin, zhangmingzhe, zhanghao}@ict.ac.cn

Abstract

Simulation is an important method to evaluate future computer systems. However, the increasing complexity of the target systems has made the development of simulators very difficult. Furthermore, detailed simulation of large-scale parallel architecture is so slow that full evaluation of real application becomes a great challenge.

This paper presents SimICT, a fast and flexible simulation framework which aims at performance and power evaluation for large-scale architecture. SimICT uses component-based design to improve its flexibility of building target systems. It also introduces an automatic parallel mechanism with relaxed synchronization to speed up the simulation. Finally, it provides a graphic configuration interface to ease the use difficulty. Based on this framework, various existing models, such as performance and power modeling tools, can be integrated to produce a holistic simulation platform.

Keywords

Framework, Parallel Simulation, Power Evaluation

1. Introduction

1.1. Motivation

Software simulator is an essential system-level design tool for both architecture design and software development. With the simulators, software developers can validate their programs without the need of real target machines, which significantly shortens design turnaround time. Also, the transparency and debuggability of the simulator can help developers quickly converge on design problems. After years of development, the single-core simulator is close to be ideal, i.e. accurate and fast. However, parallel architecture such as multi-core/many-core gradually replaces the traditional single-core architecture due to the advance in semiconductor manufacturing process. As a result, parallel architecture simulator is becoming more and more important.

1.2. Problem

However, building a large-scale parallel system simulator is notoriously difficult. The increasing complexity of both the simulated systems and target workloads has aggravated the difficulties of the development of simulators. In addition to the performance, power consumption has also become a major challenge to large-scale architecture design. Under such a situation, researchers need to improve their infrastructures and methodology to build a holistic simulator for their architectural design jobs.

Among various considerable factors of simulators, speed and flexibility are two of the most important, especially when we simulate large-scale target systems. First, as the simulated parallel system is still scaling out and the design space is becoming larger and larger, we must provide a flexible method to decrease the complexity of constructing target system and make the simulated architecture easy to be changed. Furthermore, simulation speed is also critical in this situation. Slow speed makes the evaluation of real applications impossible, so we must try to improve the simulation speed as possible as we can. Obviously, parallel simulation is the efficient method. However, it would be a very tough work to develop a parallel simulator, even for advanced engineer. In addition to that, simulator often needs to be modified to model a new target system. So the parallelization optimization specific to one target system may not work on others. The repeated work is usually boring and time consuming.

1.3. Solution

In this paper, we proposed a fast and flexible simulation framework named SimICT to cope with the problems above. The basic principle is to divide the simulator into two loosely coupled parts: the components and framework. The key features of SimICT include:

(1) Flexible simulation based on instantiated components. As we know, most of the physical computer systems are based on standard modules. So users can construct different systems by choosing different modules. Inspired by this, we can also use a component-based method to construct the target system. For a homogeneous system, we can easily simulate it by instantiating multiple components from the same class. Besides, the standard interface of components makes power models easy to be integrated.

(2) Automatic parallel mechanism with self-defined relaxed synchronization. Once the components are instantiated and the topology is defined. SimICT will schedule the components into different threads or processes and run them in parallel automatically. In this way, developers are alleviated from the tough work of making parallelization to speed up the simulator. Besides, SimICT allows the user to relax the synchronization time in order to get a better performance. This is quite important when the simulation case is very time-consuming.

(3) Visualized configurator. To ease the use difficulty, a graphic user interface (GUI) is integrated into SimICT. It can be used to instantiate the components, initialize the parameters, define the topology, and configure the SimICT. This interface is much easier to use compared with other commonly used methods like script language.

The rest of this paper is organized as follows: Section 2 discusses related work. We introduce the design of SimICT framework in Section 3. Section 4 presents some examples of using SimICT for network-on-chip performance and power evaluation. Finally, we summarize the paper in Section 5.

2. Related Work

Intel's Asim[2] is a modular framework for complexity computer system simulation. Asim copes with the complexities through modularity and reusability. Modularity helps break down the performance-modeling problem into individual pieces that can be modeled separately, while reusability allows using a software component repeatedly in different contexts. Reusability increases productivity and confidence in the robustness of the software component itself. However, Asim does not support parallel simulation.

SST [5] is an open-source, modular, parallel simulation framework. It includes a number of processor, memory, and network models. The SST has been used in a variety of network, memory, and application studies and aims to become the standard simulation framework for designing and procuring HPC system. It takes a modular approach that allows system models to be built using the components in its repository. For parallel simulation, it uses barriers to address the synchronization problem. Unlike our implementation, this is a relatively conservative algorithm.

The Manifold [7] project is an open source software project whose goal is to provide a scalable infrastructure for modeling and simulation of many core architectures. The underlying philosophy is to achieve scalable simulation capacity by using well known and tested parallel discrete event simulation (PDES) [8] algorithm and parallel time stepped simulation techniques to enable coarse grain parallel simulation of many core architectures. Manifold has good support for parallelization, but it does not address the relaxed synchronization.

SIMFLEX [14] is a simulation framework which uses component-based design and rigorous statistical sampling to enable development of complex models and ensure representative measurement results with fast simulation turnaround. The novelty of SIMFLEX lies in its combination of a unique, compile-time approach to component interconnection and a methodology for obtaining accurate results from sampled simulations on a platform capable of evaluating unmodified commercial workloads. SIMFLEX uses sampled-based method to improve the speed, not parallelization mechanism.

Simics [3] is a commercial simulator framework which has been very extensively used in both academia and industry. It has a large library of simulated components available for users to use to construct system modeling. Examples include the processors, memories, and system controllers. Various buses and networks are available. Simics also offers a convenient software test and debug environment, with features like instant stop of execution, checkpoint and restart, as well as reverse debugging and reverse execution. In order to improve the simulation speed, Simics provides a tool named Simics Accelerator [4]. It takes advantage of multicore hosts to improve the execution speed of large target system simulation. However, shared-memory multi-core target system is not well supported in this parallel mechanism.

Similar with the simulators mentioned above, SimICT also uses a component-based method to construct the target systems. However, SimICT has made some remarkable improvements. For example, when simulating a large-scale homogeneous system, we just define a component class, and construct the target system by instantiating multiple component objects through the GUI. In addition, SimICT provides a flexible synchronization mechanism which can be relaxed by the user.

There are lots of other simulators aiming at large-scale system evaluation, typical examples include Graphite[9], Hornet[10], GEMS[12], GEM5[11], COTSon[13], etc. However, these simulators either do not use component-based method, losing high flexibility, or do not support parallelization, leading to very slow simulation speed.

3. Design of SimICT

In this section, we describe the architecture and implementation details of SimICT simulation framework. We first describe the basic architecture in section 3.1. Then we introduce the implementation of framework and components in section 3.2 and section 3.3 respectively. Finally, we present the power models of SimICT in section 3.4.

3.1. Basic Architecture

The first principle of SimICT is detaching the behavior of target systems and the service of simulation. Figure 1 shows the basic architecture. As we can see, the simulator based on SimICT is composed of two parts: components and framework.

In SimICT, the simulated modules of target system, such as CPU cores, memories, routers, are organized as independent components. The Components only implement the behavior (function or timing level) of corresponding target system modules. To keep the components independent of the framework, components use standard interface to communicate with each other. Thus the simulation framework can conveniently execute the scheduling and synchronization operations. Besides, this also makes the components easy to be replaced.

Framework part is the key of SimICT. We introduce a concept of framework service (FS) to organize and schedule the components. FS is the basic scheduling unit of SimICT. Each component has to be mapped into one FS, while one FS can manage multiple components.

Figure 2 illustrates a layered view of SimICT-based simulator. The top one is user layer which contains the application and system software running on the target system. The second one is target layer consisting of all the components. Each component is responding to an individual module of the simulated systems. The third layer is simulation service layer, which mainly composes of multiple framework

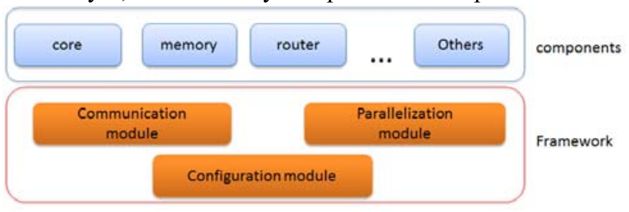


Figure 1. SimICT Architecture

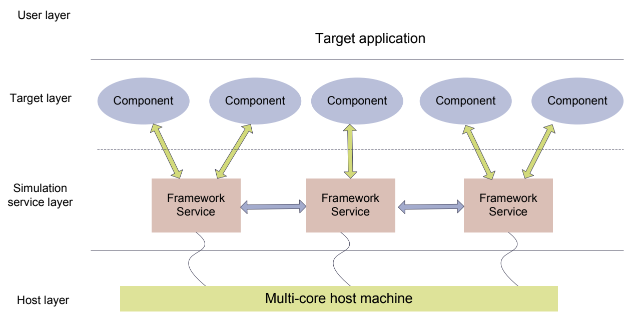


Figure 2. A Layered View of SimICT-Based Simulator

services. As we can see, framework services can communicate with each other, however, each component in target layer is isolated with others, and it completes communication operations with the help of framework services. We will discuss this in the section 3.2.1. The nethermost one is host layer, which contains the physical machine running SimICT simulator. The host is usually a multi-core system when running it in parallel.

3.2. Framework Service

As show in figure 1, there are mainly three modules in SimICT framework: communication module, parallelization module, and configuration module.

3.2.1. Communication

To keep the interfaces between components cleaner and better defined, SimICT uses a port-based communication paradigm. There are two main interface functions: port_in and port_out. When a component wants to communicate with others, it sends out the message through port_out function, with the destination's identity number and latency information in the parameters.

As we mentioned before, the components are mapped into different framework services, and SimICT schedules the framework services running on different threads or processes. Therefore, if the receiving component of port_out and the sending component locate in the same framework service, the message can be forwarded to the receiver directly. Otherwise, if the sender and receiver are in different framework services, an inter-thread or inter-process method must be used.

In order to reduce the overhead of framework service communication, SimICT executes the message sending operations together at synchronization points. For this purpose, we define an out_queue in each framework service. All the port_out operations will be buffered in the out_queue and sent out when arriving at the synchronization point. Each framework service also has an in_queue to receive the messages. All the messages in the in_queue will be sorted by processing time. When the processing time has arrived, the message is handled by calling the receiving component's port_in function.

Figure 3 shows the process of port-based communication between different framework services. The parameters in the port_out function will indicate the message's destination and latency time. The framework is responsible to send the message to the right target component at the right time.

Port-based communication gives a good privacy to each component. Component's inner state is invisible to others.

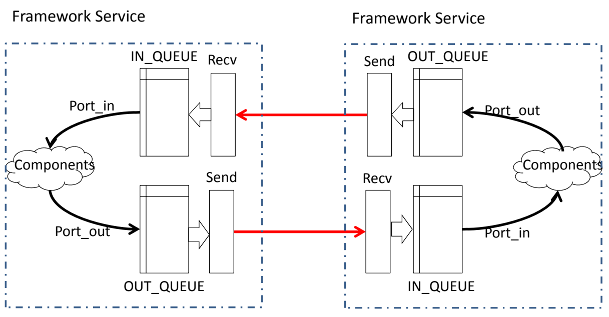


Figure 3. Port-Based Communication

Thanks to this, SimICT is able to schedule lots of components in parallel and easily change some components without affecting the others.

Unfortunately, this well-defined isolation produces a negative effect on the performance. Because all the component communication must go through the framework, it takes a relatively long time before completion. In order to avoid some unnecessary overhead, SimICT framework also provides another alternative communication mechanism based on callback functions.

If callback mechanism is used, the receiving component needs to define a function and register it as callback in the framework. Then the sending component can directly call this callback function to complete the communication. Callback function communication is faster than port-based method. However, it violates the isolation principle of components. Therefore, SimICT allows components to use callback function if it does not affect the parallelization. For example, the communication is between two components belonging to the same framework service, or the callback function does not cause any data race.

3.2.2. Parallelization

Auto-parallelization is one of the key features of SimICT. Based on port communication, each component is bound to a framework service. As shown in figure 2, SimICT assigns the framework services into different host cores, thus components belonging to different framework services can run in parallel.

This parallelization is transparent to components and does not affect the implementation of target system simulation. If the couple of FSs belong to different processes, SimICT uses MPI-style communication. Otherwise, if they belong to different threads, Pthread-style is used.

SimICT uses event-driven mode for the timing advance. For the purpose of parallelization, each framework service has an independent event queue. The FS can advance its time freely before arriving at next synchronization point. The synchronization algorithm is a key factor which affects the final performance. For SimICT, it supports two kinds of synchronization mechanism: CMB (Chandy-Misra-Bryant) algorithm [1] and quantum-based synchronization.

Strict synchronization produces accurate simulation results. However, it leads to very heavy synchronization overhead, which will further decrease the speedup results. In order to avoid this problem, SimICT lets the user to balance the speed and accuracy. It allows a user-adjusted relaxed synchronization. For example, a cycle accurate quantum-based synchronization usually forces the quantum to be as

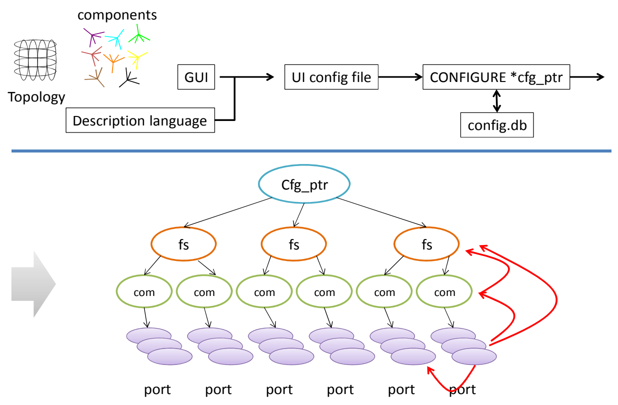


Figure 4. Configuration Flow in SimICT Framework

small as possible.

Normally a large synchronization granularity leads to a better speed, but worse accuracy. However, sometimes it is necessary to improve the speed at the cost of accuracy. Thus SimICT allows users to relax the synchronization granularity for both CMB algorithm and quantum-based method.

Relaxed synchronization means longer synchronization period, which usually incurs larger accuracy errors. Note that the functional correctness is still guaranteed as long as the benchmarks themselves are synchronized correctly.

3.2.3. Configuration

SimICT provides a powerful component-based system construction method. However, it is still a complicated task to configure a very large-scale target system. For example, if we want to simulate many-core architecture with thousands of cores, we need to instantiate thousands of core components and connect all of them according to the topology. Besides, the parameters of each core need to be set correctly. Obviously this is a very boring and fallible process.

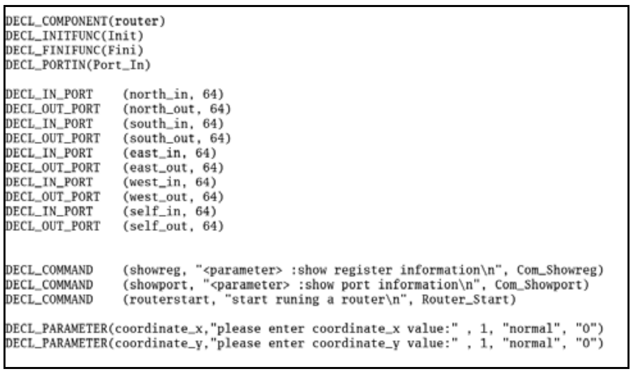
To cope with this problem, SimICT develops a graphic user interface for the system configuration, with no need for the use of obscure script languages. The flow of configuration is shown in figure 4. First, the user instantiates the components by drag-and-drop operations, and sets the topology structure by drawing the connecting lines. After that, SimICT will convert the configuration information into a record file with intermediate format. When the simulator starts, it reads in the record file, and saves the information into a structure which can be found by a pointer *cfg_ptr*.

The parameters of each component and the mapping relationship between components and framework services can also be configured through this graphic interface. SimICT stores all the configuration information into a binary database file, and loads it directly next time if the target system is the same, avoiding the reconfiguration process.

SimICT organizes the configuration information into a tree structure. It can find the position of each port, component, and framework service according to the identity number. This is necessary when handling the communication between different components.

3.3. Component

SimICT framework provides an infrastructure of the



```
DECL_COMPONENT(router)
DECL_INITFUNC(Init)
DECL_FINIFUNC(Fini)
DECL_PORTIN(Port_In)

DECL_IN_PORT    (north_in, 64)
DECL_OUT_PORT   (north_out, 64)
DECL_IN_PORT    (south_in, 64)
DECL_OUT_PORT   (south_out, 64)
DECL_IN_PORT    (east_in, 64)
DECL_OUT_PORT   (east_out, 64)
DECL_IN_PORT    (west_in, 64)
DECL_OUT_PORT   (west_out, 64)
DECL_IN_PORT    (self_in, 64)
DECL_OUT_PORT   (self_out, 64)


DECL_COMMAND    (showreg, "<parameter> :show register information\n", Com_Showreg)
DECL_COMMAND    (showport, "<parameter> :show port information\n", Com_Showport)
DECL_COMMAND    (routerstart, "start runing a router\n", Router_Start)

DECL_PARAMETER(coordinate_x,"please enter coordinate_x value:" , 1, "normal", "0")
DECL_PARAMETER(coordinate_y,"please enter coordinate_y value:" , 1, "normal", "0")
```

Figure 5. Example of Router Component Definition

simulator. We still need to implement all the components of target system. Simulating a large-scale parallel system may contain hundreds or thousands of components. Therefore, it is an arduous work to write all the components one by one. Fortunately, most large-scale systems are homogeneous, or partly homogeneous. Thus we only need to write one copy of the code for each type of component. Multiple components can be instantiated by the user in the graphic interface.

Thus, each component's code is compiled into a dynamic link library file. Some commonly used nodes, such as core, memory, memory controller, and router, exist in SimICT as dynamic link libraries. Note that, nodes with different architecture still need to be implemented independently. For example, there are different dynamic link library files for x86 core and ARM core.

SimICT uses standard interface for components communication. Furthermore, SimICT makes the definition of a new component class quite simple. Figure 5 shows an example of defining a router component class. SimICT provides some macros as template. Therefore, the user can easily finish the implementation based on these macros. The main contents include the name, initial function, finished function, in ports, out ports, command functions, and parameters.

3.4. Power Model

As an infrastructure, SimICT supports both full-system simulation and application level simulation. It also supports both performance and power evaluation.

When evaluating the power, the users can develop the new power components. Besides, SimICT also provides a standard interface to integrate other existing power modeling tools, such as Wattch [16], Orion [15], and McPAT [18]. These tools are implemented as power libraries and independent of the front-end performance components.

It should be noticed that, in addition to the performance and power models, other modeling tools can also be integrated into SimICT. For example, thermal models like HotSpot [17], and other processor models like Qsim [19] are all compatible with SimICT framework.

4. Example Study

The SimICT framework has been used for study in a several areas, including multi-core architecture, network-on-chip (NoC), power/thermal modeling, application analysis and optimization. In this section, we will give an example of NoC

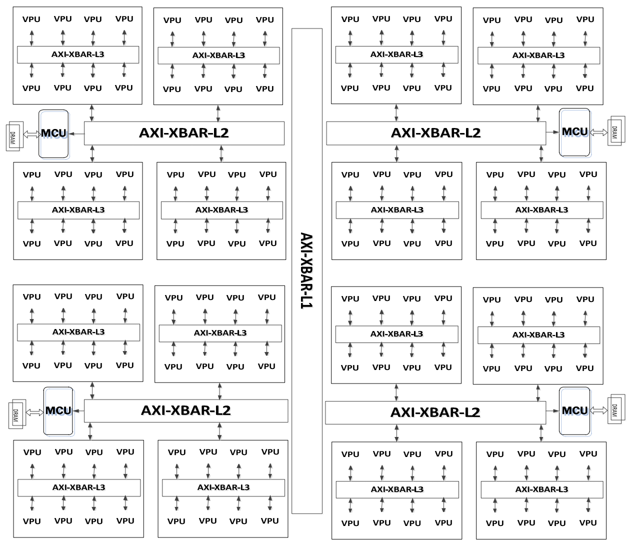


Figure 6. AXI Network Topology

simulation for performance and power analysis to demonstrate the usage of SimICT.

4.1. NoC Simulation

We setup a timing model of multi-level AXI network based on SimICT framework. The goal of this simulation is to evaluate the performance and power consumption of NoC. Figure 6 presents the topology of the network. It consists of 256 cores and three-level crossbars. Memory controllers are connected to level-two crossbar, while every eight cores are organized into a group and connected by level-three crossbar.

In the experiment, simulation is driven by read/write operations in trace file. The trace information is converted to network packets when injecting into the network. Each crossbar receives the packets, and simulates the channels of AXI protocol in detail. If the channel is available, the packet is transmitted to the next level. Otherwise if the channel is busy, it blocks the packet and forwards the event after a configured latency.

To model such a network with SimICT framework, we first define three component classes for cores, crossbars, and memory controller units respectively. Then we create multiple components by instantiating them from the classes. After that, we set the topology structure and initialize the parameters for each component through the graphic configuration interface. If we want to simulate the network in parallel, we need to set the number of threads we want to launch and the mapping relationship between components and threads. These are all the tasks we need to do before running the simulator. Obviously it is much simpler than starting from scratch.

	Parameters	Configuration
Target NoC	clock freq.	1GHz
	technology	65nm
	architecture	AXI protocol, 5 channels, 64-bit address bus, 128-bit data bus, 2-cycle latency, round-robin arbitration
Benchmark	WordCount	Input size: 100MB
	TeraSort	Input size: 100MB
	Search	20MB index file

Table 1. Evaluation Configuration

	Search	WordCount	TeraSort
Real bandwidth (Gb)	177	205	22.67
Bandwidth ratio (%)	0.82	0.95	0.11
Throughput (G packet/s)	1.23	1.43	0.16
Average response time (cycle)	3.33	17.49	43.09
Average latency (cycle)	34.83	77.17	94.99
Link usage ratio (%)	2.81	3.32	0.37
Waiting time of each packet (cycle)	2.68	7.40	10.35

Table 2. Results of Performance Evaluation

4.2. NoC Performance Evaluation

We choose threeMapReduce applications as benchmarks: Search, WordCount, and TeraSort. WordCount and TeraSort are ported form Phoenix++ [6], and Search is from Xapian project [20]. The detailed configuration of our evaluation is shown in table 1.

Table 2 gives some performance results for each benchmark. Real bandwidth, bandwidth ratio, and throughput are used to evaluate the overall performance of this network, while the remaining parameters are used to depict the detailed characteristic. We can see that, compared with Search and WordCount, TeraSort gets an obvious performance gap when running on this kind of network. This is because TeraSort has much more burst memory access operations. Besides, all the benchmarks produce a low bandwidth utilization ratio. Thus we need to take a close-up view of the detailed results. We see that normally it is not the maximum physical bandwidth that limits the performance. What we really need to do is to improve the effective utilization of the bandwidth resource.

4.3. NoC Power Evaluation

To evaluate power consumption of network, we have integrated Orion [15] into SimICT framework. To keep the flexibility, Orion is ported independently of the NoC implementation, thus this power evaluation models can also be used with other network components together. Parameters collected in runtime from network components are passed into Orion power model to produce the power evaluation results. When providing power consumption information, the Orion power model is enabled.

To evaluate the power of AXI network structure in Figure 6, Table 3 shows the results of power evaluation. We can see that power consumption has much to do with the application characteristic. Among the three benchmarks, the lowest power is only a little more than 1/4 of the highest one. The results also show that the power consumption is almost linear with the packet injection ratio. If fewer packets are injected into the network, less power would be consumed. For the similar reason, each crossbar in the lower level consumes larger power. For example, there is only one crossbar in level one. However, it consumes more than 10% power for all the three benchmarks.

	Search	WordCount	TeraSort
Level 1 (1 crossbar)	2.267	3.494	0.694
Level 2 (4 crossbars)	6.484	9.182	1.828
Level 3 (16 crossbars)	4.484	7.430	3.166
Total	13.235	20.106	5.688

Table 3. Results of Power Evaluation (W)

5. Conclusion and Future Work

This paper presents SimICT, a fast and flexible framework which aims at performance and power evaluation for large-scale architecture. There are three key features of this framework: First, SimICT provides a flexible simulation based on instantiated components. The standard interface of components also makes power models easy to be integrated. Second, it implements an automatic parallel mechanism with self-defined relaxed synchronization, producing better performance for large-scale system simulation. Finally, a visualized configurator is integrated into SimICT, making it much easier to use compared with other commonly used methods like script languages.

Based on SimICT, various existing models, such as performance and power modeling tools, can be integrated to produce a holistic simulation platform. We believe that SimICT framework could be an attractive option for achieving fast and flexible simulation for large-scale parallel architecture.

There are mainly two directions in our future work. First, we will continue to optimize the communication and synchronization of SimICT to further improve the simulation speed. Second, we are going to develop more components and integrate more existing models, such as thermal and reliability tools, into SimICT.

Acknowledgments

This work is supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302501, the National Science Foundation for Distinguished Young Scholars of China under Grant No. 60925009, the Foundation for Innovative Research Groups of the National Natural Science Foundation of China under Grant No. 60921002, the Beijing science and technology plans under Grant No.2010B058 and the National Natural Science Foundation of China under Grant No.(61173007, 61100013, 61100015, 61204047 and 61202059), the National High-Tech Research & Development Program of China（2012AA010303）

References

1. K.M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering, SE-5(5):440–452, Sept. 1979.
2. J. Emer, P. Ahuja, E. Borch, A. Klauser, Chi-Keung Luk, S. Manne, S. S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan. Asim: A performance model framework. IEEE Computer, 35(2):68 – 76, February 2002.
3. P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. Computer, 35:50–58, 2002.
4. J. Engblom. Simics Accelerator. VIRTUTECH White Paper. 2009.
5. A.F.Rodrigues. The structural simulation toolkit, http:// www.cs.sandia.gov/sst, 2007
6. Justin Talbot, Richard M. Yoo, Christos Kozyrakis. Phoenix++: Modular MapReduce for Shared-Memory Systems. Second International Workshop on MapReduce and its Applications (MAPREDUCE) 2011.
7. Manifold Project. http://manifold.gatech.edu.
8. R.M. Fujimoto. Parallel discrete event simulation. Commun. ACM, 33(10):30–53, 1990.
9. J. 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 Proc. HPCA, 2010.
10. M. Lis, P. Ren, M. Cho, K. Shim, C. Fletcher, O.Khan, and S. Devadas. Scalable, accurate multicore simulation in the 1000-core era. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), April 2011.
11. N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, , and D. A. Wood. The gem5 simulator. Computer Architecture News, 2011.
12. Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, David A. Wood: Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. SIGARCH Computer Architecture News 33(4): 92-99 (2005)
13. E. Argollo, A. Falc´on, 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
14. Nikolaos Hardavellas, Stephen Somogyi, Thomas F. Wenisch, Roland E. Wunderlich, Shelley Chen, Jangwoo Kim, Babak Falsafi, James C. Hoe, Andreas Nowatzyk: SimFlex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. SIGMETRICS Performance Evaluation Review 31(4): 31-34 (2004)
15. S. Singh, C. May_eld, S. Mittal, S. Prabhakar, S. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. 2008 ACM SIGMOD Int. Conf. Management of Data. ACM, NY, 2008, pp. 1239-1242.
16. D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. 27th Int. Sym. Computer Architecture (ISCA '00). ACM, NY, 2000, pp. 83-94.
17. K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. ACM Trans. Archit. Code Optim. 1, 1 (Mar. 2004), pp. 94-125.
18. S. Li et al. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. 42nd Ann. IEEE/ACM Int. Sym. Microarchitecture, 469{480, 2009.
19. C. Kersey, A. Rodrigues, and S. Yalamanchili. A universal parallel front-end for execution driven microarchitecture simulation. Proceedings of the 2012 Workshop on Rapid Simulation and Performance Evaluation Methods and Tools, pages 25 – 32, 2012.
20. http://xapian.org/