International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 2, pages 217–227, July 2013

Naxim: A Fast and Retargetable Network-on-Chip Simulator with QEMU and SystemC

Keita Nakajima
Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Shuto Kurebayashi Yusuke Fukutsuka Takuji Hieda
Ittetsu Taniguchi Hiroyuki Tomiyama
College of Science and Engineering, Ritsumeikan University
1-1-1 Noji-Higashi, Kusatsu, Shiga 525-8603, Japan

Hiroaki Takada
Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8603, Japan

Received: February 19, 2013
Revised: May 21, 2013
Accepted: June 18, 2013
Communicated by Yasuaki Ito

Abstract

Systems-on-Chip (SoC) architectures have been shifting from single-core to multi-core solutions, and they are at present evolving towards many-core ones. Network-on-Chip (NoC) is considered as a promising interconnection scheme for many-core SoCs since it offers better scalability than traditional bus-based interconnection. In this work, we have developed a fast simulator of NoC architectures using QEMU and SystemC. QEMU is an open-source CPU emulator which is widely used in many simulation platforms such as Android Emulator. In the proposed simulator, each CPU core is emulated by a QEMU, and the network part including NoC routers is modeled with SystemC. The SystemC simulator and QEMUs are connected by TCP sockets on a host computer. Our simulator is fast because QEMUs run in parallel on a multi-core host computer or even multiple host computers. Also, our simulator is highly retargetable because QEMU provides a variety of CPU models and we use QEMU as is. In our experiments, our simulator successfully simulates a 108-core NoC in a practical time. We have also confirmed the scalability and retargetability of our NoC simulator.

Keywords: instruction-set simulation, network-on-chip, SystemC, software development support

1 Introduction

The continuous advance in semiconductor technology enables us to place more CPU cores on a single System-on-Chip (SoC). According to ITRS Report 2011 Edition [1], the number of cores on a chip has been increasing by 40% every year, and it will exceed a hundred cores in 2017 for high-performance embedded applications such as networking routers. In the forthcoming many-core SoC era, the traditional bus-based interconnection architecture is no longer feasible due to its poor throughput performance. Network-on-Chip (NoC), in which cores communicate with each other by sending/receiving packets through routers (switches), has emerged as a promising alternative paradigm for on-chip communication. Compared with the traditional bus-based interconnection, NoC brings various benefits such as shorter wire delay, higher overall throughput, higher fault tolerance, regularity in physical design, and so on.

Along with an increase in the number of cores, software development as well as hardware design becomes more complicated and time-consuming. In many cases of embedded software development, what is worse, hardware prototype boards are not available until very late stages of the system design. Before the real hardware becomes available, software development inevitably relies on simulation on host computers. Software simulation is broadly classified into two methods [2]. One is native execution (host-compiled simulation in other words), in which software is directly compiled with a native compiler and executed on the host computer. Native execution is very fast, but its execution behavior on the host computer may be very different from that on the target computer. Another drawback of native execution is that it cannot be used for development of Hardware-dependent Software (HdS). The other method is interpretive simulation, in which software is cross-compiled with a compiler of the target CPU, and the compiled code is executed on an instruction-set simulator of the target hardware. Interpretive simulation behaves more accurately than native execution, but suffers from a lower execution speed. Thus, the two methods are complementary. At early phases of software development, software functionality is extensively validated by means of native execution, and at later phases, the software is carefully optimized and tested by interpretive simulation. This paper studies interpretive software simulation for NoC architectures.

Several innovative techniques have been developed in the past for the improved speed of interpretive software simulation. Among them, dynamic binary translation is one of the most efficient techniques, and is employed by a number of simulators and virtual machines developed in industry and academia. QEMU is one of the most popular simulators with dynamic binary translation [3]. QEMU supports a variety of processors including multicore processors, but it does not support NoC-based many-core processors by itself.

This paper describes a simulator, named *Naxim*, for NoC-based many-core architectures using QEMU and SystemC. SystemC is a de-facto standard language for event-driven simulation of electronic systems, and its reference simulator is freely available from the website of Accellera [4]. The Naxim simulator uses a SystemC simulator and a set of QEMUs. Each CPU core is simulated by a QEMU. In case of N-core NoC, therefore, N QEMUs are executed. The network part including NoC routers is modeled with SystemC. The SystemC simulator and QEMUs are connected by the TCP socket on a host computer.

The Naxim simulator features as follows.

- The Naxim simulator is efficiently executed on a multicore host computer. Recent processors used in host computers (i.e., PC and workstations) have multiple CPU cores, over which multiple threads/applications are executed in parallel. In Naxim, the SystemC simulator and individual QEMUs are different applications from a viewpoint of the host OS. Thus, the SystemC simulator and QEMUs can be executed in parallel on the host computer.
- The Naxim simulator can be executed on multiple host computers since the SystemC simulator and QEMUs are connected via standard TCP sockets.
- The Naxim simulator is highly retargetable since QEMU supports a variety of processors and we use QEMU without any modification.
- The Naxim simulator consists of SystemC and QEMU, both of which are open-source software. Commercial software is not used in our simulator.

The reminder of this paper is organized as follows. Section 2 reviews related works. Section 3 describes the Naxim simulator, and Section 4 shows a set of experiments and discusses the effectiveness of our simulator. Finally, Section 5 concludes this paper with a summary.

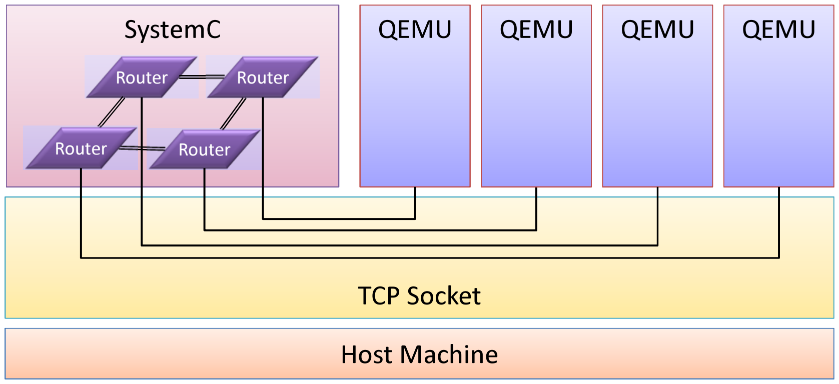

Figure 1: An overview of the Naxim simulator.

2 Related Work

In the past, a number of NoC simulators have been developed in the world. C++, Java and SystemC are popular languages to develop NoC simulators. NoC simulators written in SystemC include Noxim [5], NNSE [6] and PPNOCS [7]. Most of existing NoC simulators simulate on-chip interconnection network only, and do not simulate CPU cores. They are suitable to evaluate the performance of network architectures using synthetic packets, but are not applicable to software development for NoCs. On the contrary, our NoC simulator simulates CPU cores as well as on-chip interconnection network, and thus is applicable to software development.

In [8], the authors have devolped a hardware/software cosimulator using QEMU and SystemC. Their simulator has cycle-level timing accuracy, but does not support multi/many-core SoCs.

Several researchers have developed simulation platforms which take advantage of QEMU and SystemC. The core of the simulation platforms is a set of libraries to connect QEMU and SystemC so that users can easily devlop their own simulators. Examples of such simulation platforms include QBox [9, 10], TLMu [11], and Rabbits [12, 13]. Some of them support simulation of multi-core architectures, but to the best of our knowledge, they are not used for simulation of NoCs with more than 100 cores. Our NoC simulator, on the other hand, can successfully simulate 108-core NoCs at a practical speed.

There also exist several simulation platforms which are not based on QEMU and SystemC. Examples of such simulation platforms include GEMS [14] and GEM5 [15]. GEMS is often used in conjunction with Wind River Simics [16, 17]. Using these platforms, engineers can easily devlop a simulator of their own NoC architecture. To our understanding, however, these simulators are executed as a single process on a host computer, and thus, cannot take advantage of the multi-core parallelism of the host computer.

3 The Naxim Simulator

This section describes an overview and detailed organization of our NoC simulator.

3.1 Overview

At present, our NoC simulator is executable on Linux-based host computers. However, it can be easily portable to different OS machines which support TCP/IP protocols.

Figure 1 shows an overview of our simulator for a quad-core NoC architecture. The simulator consists of an OSCI SystemC reference simulator and four QEMUs. Each QEMU simulates a CPU core. The network part including routers and interconnection between the routers are modeled in SystemC, and executed on the SystemC simulator. Each router module modeled in SystemC is

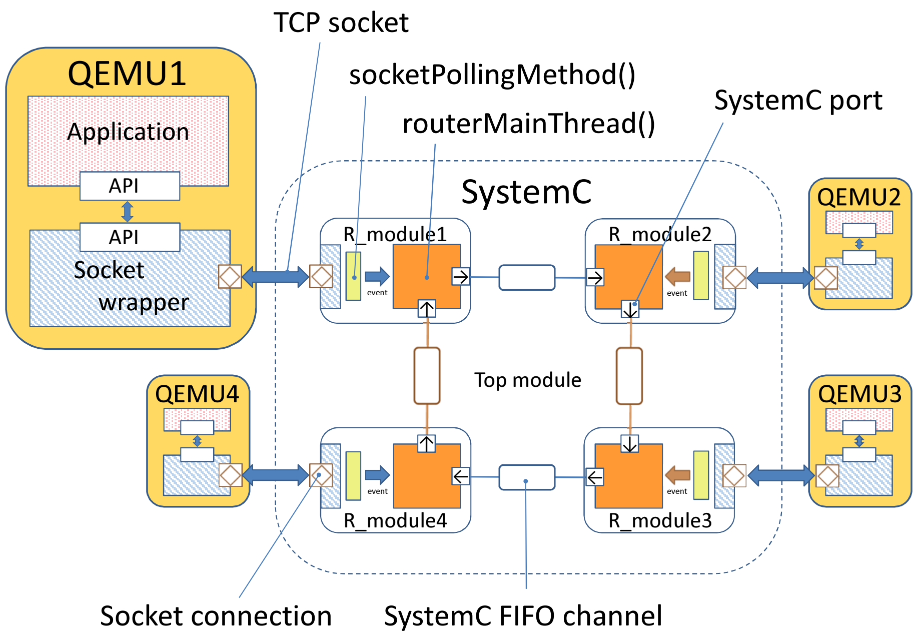

Figure 2: Internal organization of the Naxim simulator.

connected to a corresponding QEMU by a TCP socket. The SystemC simulator and QEMUs are executed as different processes on a host machine.

NoC topologies supported by our simulator are ring and 2D-mesh. At present, the NoC router in Naxim is very simplified because Naxim is developed mainly for software designers who want to perform functional debugging of software and do not require details of the hardware architecture. Our simple router model features static X-Y routing, store-and-forward switching, no virtual channel, unlimited-size buffer for each direction, no pipelining, and delta-time latency. These restrictions do not come from our simulation platform, but from IPs of router modules. Our simulator can simulate a wider range of NoC architectures by developing new router modules in SystemC.

The timing accuracy of Naxim is not cycle-accurate, and thus, Naxim cannot used for performance evaluation of software or hardware. More specifically, neither QEMU nor the network part in SystemC is cycle-accurate. QEMU counts the number of instructions executed, but does not count the execution cycles [1]. The SystemC part is not cycle-accurate, either. A packet is transmitted from router to router in a *delta* time-unit. This means that it takes zero clock cycle to transmit a packet from source core to destination one.

3.2 Communication between QEMU and SystemC

Figure 2 shows an internal organization of our simulator for a unidirectional ring-based NoC with four cores. Four routers are modeled in SystemC, each of which is connected to QEMU via a TCP socket. We decided to use TCP sockets due to their popularity and reliability. TPC sockets are supported by most of major operating systems such as Linux and MS-Windows. Therefore, porting Naxim onto different host computers is not difficult. TCP sockets are reliable in the sense that it is guaranteed that sockets are delivered in order without being lost [2].

QEMU features two types of simulation, i.e., user mode emulation and full system emulation. User mode emulation executes a single application program without running an OS on the QEMU. In other words, QEMU simulates a target processor only, and do not simulate its peripheral devices.

[1]Although original QEMU is not cycle-accurate, some research groups such as [12, 18] have studied cycle-accurate simulation based on QEMU. We can utilize their achievements at the cost of degraded portability and simulation speed.

[2]TCP sockets are reliable at the cost of performance overhead. For example, UDP sockets are faster than TPC ones, but UDP is not as reliable as TCP. UDP sockets may be delivered out of order, or may be lost in the worst case.

```
int sendPacket(const char* p, const int& len, const int& id);
// p   : pointer to data to be sent
// len : size of data to be sent (in bytes)
// id  : ID of destination core
// return value : size of data sent successfully (in bytes)

int receivePacket(const int& len, const char* p);
// len : size of data to be received (in bytes)
// p   : memory address to copy received data
// return value : size of data received successfully (in bytes)
```

Figure 3: Socket wrapper APIs.

System calls executed on the QEMU are passed to the OS of the host computer. On the other hand, full system emulation simulates the entire target computer, including not only the target processor but also its peripheral devices. An OS runs on the QEMU, and system calls from an application program are handled by the OS running on the QEMU. In either type of simulation, TCP/IP networking applications are runnable on QEMUs. In user mode simulation, network system calls are passed to the host OS, whereas the network system calls are handled by the OS on the QEMU in full system emulation. The Naxim simulator takes advantage of this feature of QEMU in order to connect QEMUs to the SystemC simulator. The Naxim simulator supports both user mode emulation and full system emulation of QEMU. We use TCP sockets as a fundamental communication protocol. Since TCP sockets are very primitive, we have developed an additional protocol layer to provide easy-to-use APIs to application programs. The layer is named *socket wrapper* in Figure 2. The socket wrapper APIs include simple send/receive functions as shown in Figure 3.

The network part of NoC is modeled in SystemC. A router is modeled as a SystemC module. As shown in Figure 2, each router module has a wrapper for TCP networking, and a method, named `socketPollingMethod()`, monitors the TCP socket at every simulation cycle. When a packet arrives from QEMU, the router sends it to one of neighbors according to the NoC routing protocol. Packet routing is handled by a SystemC thread, named `routerMainThread()` shown in Figure 2. Communication between routers is implemented by simple FIFO channels which are provided by the SystemC standard library. When a packet arrives from a neighbor router, the receiver router checks the destination of the packet. If the destination is the receiver router itself, the packet is sent to QEMU via the TCP socket. If the destination is different, the packet is passed to a neighbor router according to the routing protocol. Fragments of SystemC code for the top module and the router module are presented in Figures 4 and 5, respectively. Note that these SystemC code fragments are written for the 4-core unidirectional-ring NoC shown in Figure 2, and thus the router has only a single input port and a single output port. In case of 2D-mesh NoCs, each router has four input ports and four output ports for inter-core communication.

4 Experiments

We have conducted a set of experiments to demonstrate the effectiveness of the Naxim simulator. Specifically, we have evaluated performance scalability and retargetability of the Naxim simulator. In addition, we have compared the elapsed simulation time of Naxim with that of an existing simulator.

4.1 Experimental Setup

We used a JPEG encoding application as our benchmark program. The JPEG application was pipelined with nine stages, each of which is mapped to a CPU core. We have simulated six NoC

```
class Top : public sc_module
{
  // SystemC channel declaration
  sc_fifo<internal_packet>  ch_instance1;
  sc_fifo<internal_packet>  ch_instance2;
  sc_fifo<internal_packet>  ch_instance3;
  sc_fifo<internal_packet>  ch_instance4;

  Router  R_module1; // Router instantiation
  Router  R_module2;
  Router  R_module3;
  Router  R_module4;

  R_module1.out_port(ch_instance1);
  R_module2.in_port (ch_instance1);
  R_module2.out_port(ch_instance2);
  R_module3.in_port (ch_instance2);
  R_module3.out_port(ch_instance3);
  R_module4.in_port (ch_instance3);
  R_module4.out_port(ch_instance4);
  R_module1.in_port (ch_instance4);

  // Other modules if any
}
```

Figure 4: Part of SystemC code for the top module.

```
class Router : public sc_module
{
  // SystemC port declaration
  sc_fifo_in  in_port;
  sc_fifo_out out_port;

  sc_event arrival_event; // event of packet arrival from core

  socketWrapper sw; // instantiation of socket wrapper

  void socketPollingMethod();
  void routerMainThread();

  // user-defined threads and methods
}
```

Figure 5: Part of SystemC code for the router module.

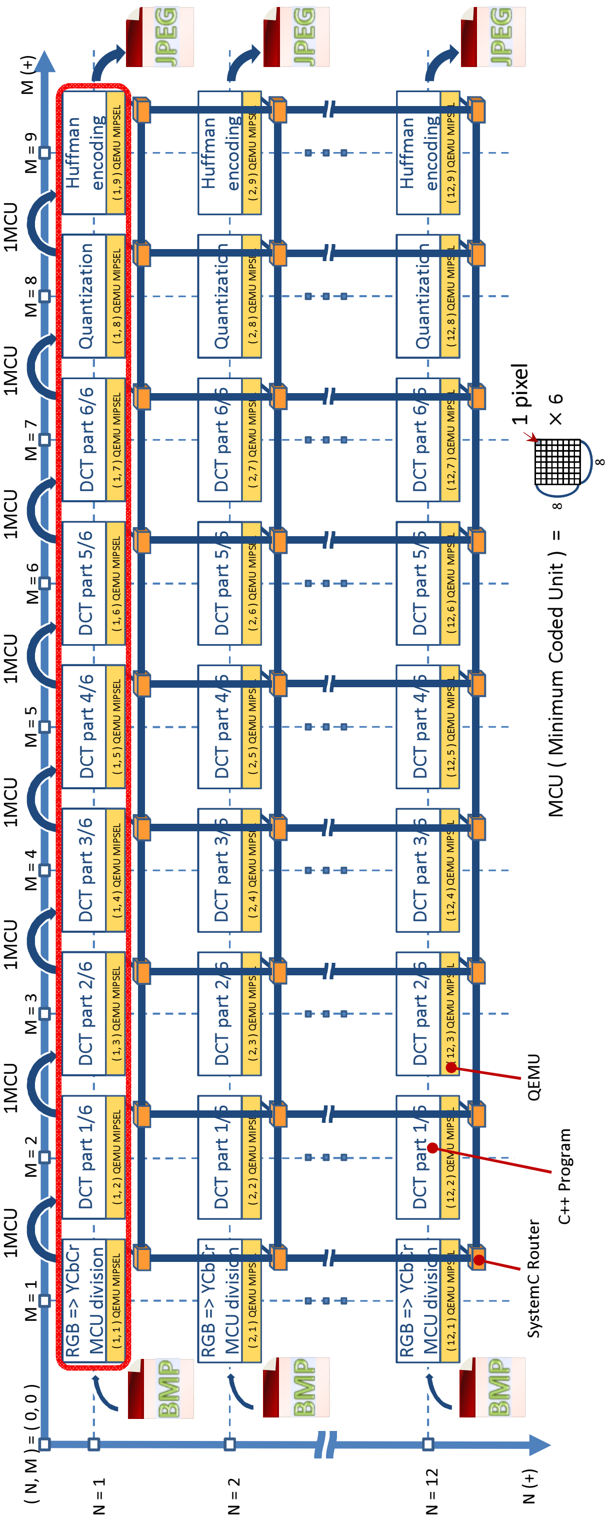

Figure 6: Mapping of JPEG applications onto NoC architectures.

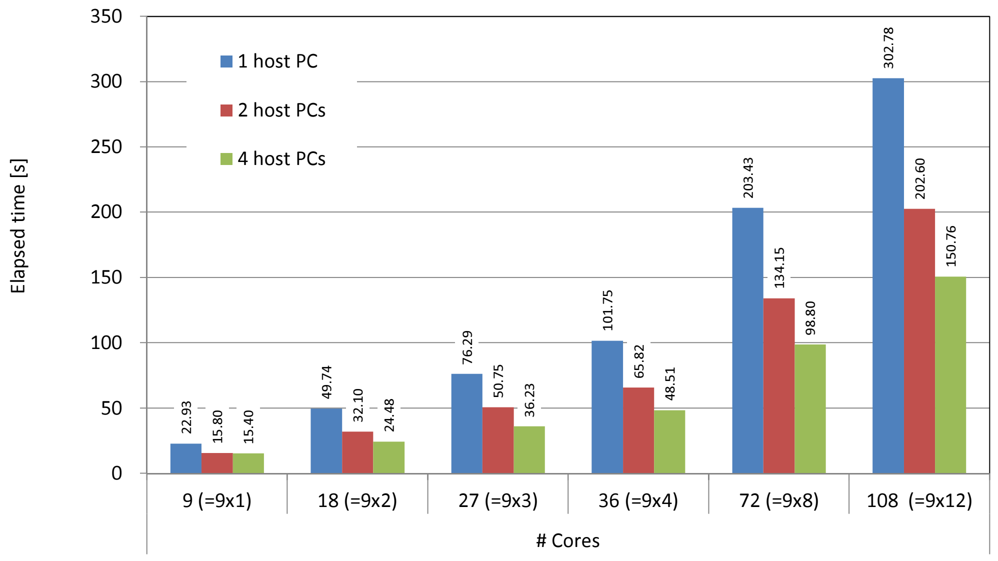

Figure 7: Elapsed simulation times for different-size NoCs.

architectures with 9, 18, 27, 36, 72 and 108 cores. The NoC architectures are based on 2D-mesh with XY routing. As shown in Figure 6, multiple JPEG applications are executed on the NoC architectures. In case of a 18-core NoC, two images are encoded in parallel.

4.2 Scalability

First, we ran the Naxim simulator on a single host computer with Intel Core2 Duo P8700 (2.53GHz, dual cores, no hyperthreading), 4GB DDR2-800 memory and Ubuntu Linux. We changed the number of cores on the NoC along with the number of image streams, and measured the simulation time. The size of images to be encoded is 1024-by-768 pixels. The ISA of the target cores is i386. The simulation time is shown in blue bars in Figure 7. The results show that the simulation time is nearly proportional to the number of cores. We successfully simulated up to 108-core NoC in a practical time. The results demonstrate an excellent scalability of Naxim.

Next, we ran the Naxim simulator on two and four host computers. All of the host computers were identical, with the same features as described above. The SystemC simulator was executed on one of the hosts, and QEMUs were evenly distributed over the hosts. The simulation times on two hosts and four hosts are shown in red and green bars, respectively, in Figure 7. The distributed simulation on the two host computers is 34% faster than the single-host simulation, and simulation on the four hosts is 48% faster. The results demonstrate another-level scalability, i.e., the scalability with respect to the number of host computers as well as the number of target cores.

4.3 Retargetability

One of the important features of our NoC simulator is its high portability. Our simulator supports any CPU core supported by QEMU since we use QEMU without modification. In order to test the retargetability of our simulator, we changed the ISA of target cores and measured the simulation times. We also ran QEMU as both user mode emulation and full system emulation. In case of full system emulation, we used Debian Linux as a target OS. A single-stream JPEG encoder was executed on a nine-core NoC architecture. The image size is 3648-by-2736 pixels. The host computer features Intel Core i7 990x (3.46GHz, 6 cores, 12 threads), 12GB DDR3-1333 memory and Ubuntu Linux. The simulation results are shown in Figure 8. In the figure, native refers to native software execution without QEMU. It is confirmed that a wide range of ISAs are supported by our simulator.

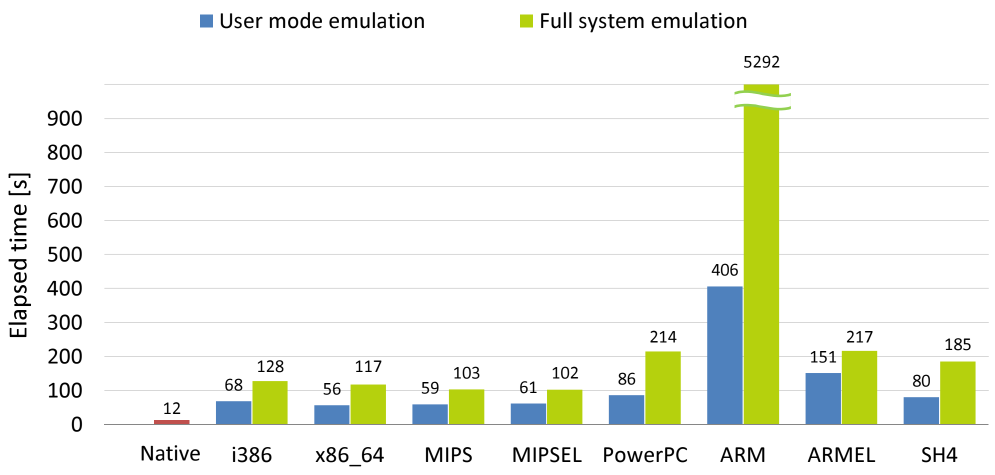

Figure 8: Elapsed simulation times for different ISAs.

Table 1: Comparison of elapsed simulation time with an existing simulator.

	Elapsed simulation time
TLMu	23,466 sec
Naxim	24 sec

4.4 Comparison with an Existing Simulator

It is very difficult to quantitatively compare the performance of Naxim with other simulators. In order to do so fairly, we need to create a simulation model of the same NoC architecture (i.e., same instruction-set architecture, same interconnection architecture, same peripherals, and so on) on a different simulation platform at the same level of abstraction, which was impossible for us.

Although it is not completely fair, we have created a simulation model of a 9-core SoC using the TLMu library [11][3] and compared its simulation time with Naxim just for a reference. The SoC modeled with TLMu is not based on NoC but a traditional bus-based architecture, but the abstraction level of on-chip communication is the same as Naxim. The ISA of the cores is ARM.

We ran the 9-stage pipelined JPEG encoder application (1024-by-768 pixels) on the two simulators as follows:

TLMu: A simulator of a bus-based 9-core SoC modeled with TLMu.

Naxim: A simulator of a 2D-mesh 9-core NoC modeled with Naxim.

Table 1 shows the elapsed simulation times of the two simulators. The results show that the elapsed simulation time of Naxim is shorter than that of TLMu by approximately three orders of magnitude. Although the modeled architectures are not identical, this result clearly demonstrates the high performance of the Naxim simulator.

5 Conclusions

In this work, we have developed a fast simulator of many-core NoC architectures, named Naxim, which takes advantage of QEMU and SystemC. QEMU simulates a CPU core, while SystemC is used for simulation of routers and interconnection. The Naxim simulator efficiently runs on a multi-core host computer and even on multiple host computers. Our experimental results demonstrate the high performance, scalability and retargetability of the Naxim simulator.

[3]More precisely, we used the TLMu library which was modified by hdLab [19].

At present, the timing of our simulator is not cycle-accurate. Improvement of the timing accuracy is a subject of our future work. Also, we plan to support a wider range of NoC routing and switching mechanisms.

Acknowledgments

We would like to thank Mr. Yosuke Kurimoto for his supports to conduct our experiments.

This work is supported in part by New Energy and Industrial Technology Development Organization (NEDO), Japan.

References

[1] ITRS 2011 Edition, http://www.itrs.net/Links/2011ITRS/Home2011.htm.

[2] F. Petrot, M. Gligor, M.-M. Hamayun, S. Hao, N. Fournel and P. Gerin, "On MPSoC Software Execution at the Transaction Level," *IEEE Design and Test of Computers*, vol. 28, no. 3, pp. 32-43, 2011.

[3] F. Bellard, "QEMU, a Fast and Portable Dynamic Translator," *USENIX Annual Technical Conference*, 2005.

[4] Accellera Systems Initiative, http://www.accellera.org/.

[5] M. Palesi, D. Patti and F. Fazzino, *Noxim: The NoC Simulator*, http://noxim.sourceforge.net/.

[6] Z. Lu, R. Thid, M. Millberg, E. Nilsson and A. Jantsch, "NNSE: Nostrum Network-on-Chip Simulation Environment," *Swedish System-on-Chip Conference*, 2005.

[7] E. S. M. Saad, S. A. Salem, M. H. Awadalla and A. M. Mostafa, "PPNOCS: Performance and Power Network on Chip Simulator based on SystemC," *International Journal of Computer Science*, vol.8, issue 6, Nov. 2011.

[8] M.-C. Chiang, T.-C. Yeh and G.-F. Tseng, "A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, Apr. 2011.

[9] M. Monton, A. Portero, M. Moreno, B. Martinez and J. Carrabina, "Mixed SW/SystemC SoC Emulation Framework," *International Symposim on Industrial Electronics*, 2007.

[10] GreenSocs Ltd., *QBox: QEMU based Full System Virtual Platforms*, http://www.greensocs.com/projects/QEMUSystemC.

[11] F. Bellard, *TLMu: Transaction Level eMulator*, http://edgarigl.github.com/tlmu/.

[12] M. Gligor, N. Fournel and F. Petrot, "Using Binary Translation in Event Driven Simulation for Fast and Flexible MPSoC Simulation," *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2009.

[13] TIMA Laboratory, *Rabbits: An Environment for Fast and Accurate MPSoC Simulation*, http://tima-sls.imag.fr/www/research/rabbits.

[14] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill and D. A. Wood, "Multifacet general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, 2005.

[15] 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, D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39 no. 2, pp. 1–7, 2011.

[16] 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," *IEEE Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[17] Wind River, http://www.windriver.com/products/simics/.

[18] D. Thach, Y Tamiya, S Kuwamura and A Ike, "Fast Cycle Estimation Methodology for Instruction-Level Emulator," *Design Automation and Test in Europe*, 2012.

[19] hdLab, http://www.hdlab.co.jp/web/a050consulting/b009armcpumodel/.