A Hardware/Software Cosimulator for Network-on-Chip

Yosuke Kurimoto¹ Yusuke Fukutsuka² Ittetsu Taniguchi² Hiroyuki Tomiyama²

¹Graduate School of Science and Engineering, Ritsumeikan University, 1-1-1 Noji-higashi, Kusatsu, Shiga 525–8577 Japan.
²College of Science and Engineering, Ritsumeikan University, 1-1-1 Noji-higashi, Kusatsu, Shiga 525–8577 Japan.
E-mail: {yosuke.kurimoto, yusuke.fukutsuka}@tomiyama-lab.org, {i-tanigu, ht}@fc.ritsumei.ac.jp

Abstract

Network-on-Chip (NoC) is considered as a promising interconnection scheme for many-core System-on-a-Chip (SoC) since it offers better scalability than traditional bus-based interconnection. In this paper, we proposed a fast simulator of NoC architectures using QEMU and Noxim. QEMU is an open-source CPU emulator, and Noxim is an open-source SystemC base NoC simulator for on-chip interconnection network. Experimental results show that proposed simulator successfully simulates a 108 core NoC in a practical time.

Keywords—instruction-set simulation; network-on-chip; SystemC; software development support

I. 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. In the forthcoming manycore SoC era, the traditional bus-based interconnection architecture is no longer feasible due to its poor throughput. 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 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 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 manycore processors by itself.

This paper describes a simulator of NoC-based manycore architectures using QEMU and Noxim. Noxim is an open-source SystemC base NoC simulator for on-chip interconnection network [4]. Our NoC simulator consists of a set of QEMUs and the Noxim. Each CPU core is simulated by a QEMU. Noxim and QEMUs are connected by TCP sockets on a host computer. Our NoC simulator features as follows.

- Our NoC 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/application are executed in parallel. In our NoC simulator, Noxim and individual QEMUs are different applications from a viewpoint of host OS. Thus, Noxim and QEMUs can be executed in parallel on the host computer.
- Our NoC simulator can be executed on multiple host computers since Noxim and QEMUs are connected via standard TCP sockets.
- Our NoC simulator is highly retargetable since QEMU supports a variety of processors and we use QEMU without any modification.
Our NoC simulator consists of Noxim and QEMU, both of which are open-source software. Commercial software is not used in our simulator.

The rest of this paper is organized as follows. Section II reviews related works, and Section III describes our NoC simulator. Section IV shows experimental results. Section V concludes this paper with a summary.

II. RELATED WORK

In the past, a number of NoC simulators have been developed in the world. C++, Java and SystemC are popular languages to developed NoC simulators. NoC simulators written in SystemC include NNSE [5], PPNOCS [6] and Noxim [4]. 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.

To realize NoC simulator for software development, we previously proposed fast NoC simulator with QEMU and SystemC [7]. Our previous simulator named “Naxim” simulates CPU core by QEMU as well as on-chip interconnection network by SystemC, and fast NoC simulation is achieved. However, the simulation accuracy of on-chip interconnection network is untimed, and this is the limitation of Naxim. Our NoC simulator simulates CPU cores by QEMUs as well as on-chip interconnection network by Noxim. Thus, cycle accurate on-chip interconnection network simulation is available.

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 develop their own simulators. Examples of such simulation platforms include QBox [8], TLMu [9], and Rabbits [10]. 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 in a practical time.

III. A HARDWARE/SOFTWARE COSIMULATOR FOR NOC

A. Overview

Figure 1 shows an overview of our simulator for a quad-core NoC architecture. The simulator consists of the Noxim developed using SystemC and four QEMUs. Each QEMU simulates each CPU core. The network part including routers and interconnection between routers are simulated by Noxim. Each PE module in Noxim is connected to a corresponding QEMU by a TCP socket. Noxim and QEMUs are executed as different processes on a host computer.

B. QEMU

QEMU is an open-source CPU emulator. 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. On the other hand, full system emulation simulates the entire target computer, including not only the target processor but also its peripheral devices. Our simulator supports both user mode emulation and full emulation of QEMU.

C. Noxim

Noxim is an open-source NoC simulator developed at the University of Catania [4]. Noxim provides flexibility to specify many different properties of NoC architecture such as network size, routing algorithm, buffer size, etc. Noxim allows NoC evaluation in terms of throughput, delay and power consumption using synthetic packets.

Figure 2 shows an overview of Noxim. The architecture model of Noxim is a tile-based NoC architecture. Each tile module includes PE module and router module. The PE module artificially generates the network traffic, and also analyzes the received packet to calculate throughput, delay and power consumption. The router module transmits the received packet to the adjacent router module. Each packet only includes the size of packet, time stamps, and IDs of source and destination. Table 1 summarizes the feature of Noxim. Changing these parameters, our NoC simulator supports various NoC architectures.

<table>
<thead>
<tr>
<th>Topology</th>
<th>2D Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing</td>
<td>Cycle Accurate</td>
</tr>
<tr>
<td>Routing</td>
<td>XY, Adaptive, West-first, etc.</td>
</tr>
<tr>
<td>Switching</td>
<td>Wormhole</td>
</tr>
<tr>
<td>Channel</td>
<td>Virtual Channel based</td>
</tr>
</tbody>
</table>
D. Inter-QEMU Packet Transmission

Since our NoC simulator simulates CPU cores by QEMU and on-chip interconnection network by Noxim, the mechanism to realize inter-QEMU packet transmission through Noxim is necessary. This section proposes our ideas to connect QEMU and Noxim, and the method to realize hardware/software cosimulation.

Figure 3 shows an internal organization of our NoC simulator for 2D-Mesh NoC with four CPU cores. CPU cores are emulated by QEMUs, and on-chip interconnection network is simulated by Noxim. We use TCP sockets as a fundamental communication protocol between QEMU and Noxim. Since TCP sockets are very primitive, we have developed an additional protocol layer to provide easy-to-use APIs include simple send/receive functions as socket wrapper.

Receiving the packet from QEMU via a TCP socket, Noxim transmits the packet to the destination. However, as explained in previous subsection, Noxim generates an artificial traffic, and generated packets do not contain any substantial data. In order to keep both cycle accurate on-chip interconnection network simulation and the substantial data transmission, we have modified the PE module in Noxim.

Figure 4 shows an overview of modified PE module in Noxim. The modified PE module includes socketPollingMethod(), rxProcess(), and txProcess() functions.

rxProcess() and txProcess() are Noxim original process, and socketPollingMethod() is our original method.

Originally txProcess() generates an artificial traffic, and the synthetic packets are divided into flits according to the packet information. Then the process sends the flits to the corresponding router. rxProcess() receives the synthetic packet from the router, and calculates throughput, delay and power consumption. Notice that the packet does not contain any substantial data.

In order to support the substantial packet transmission, we developed socketPollingMethod() and modifies Noxim original rxProcess() and txProcess(). When socketPollingMethod() receives the packet from QEMU via TCP socket, txProcess() is called to send the packet. txProcess() first put the packet itself to the packet buffer, which is our original buffer placed on each PE, in the sender PE. Then the synthetic packet with the same packet size is generated and transmitted in the same as original way.

When rxProcess() receives the synthetic packet, the sender PE ID is extracted from the received packet. Then rxProcess() receives the substantial packet from the packet buffer in sender PE, and sends the packet to destination QEMU directly via TCP socket.

IV. EXPERIMENTS

To demonstrate the efficiency of our NoC simulator, we evaluate our NoC simulator in terms of simulation time. To simulate various NoC architectures, we vary the number of cores from 18 to 108, and link bandwidth from 64bit to 256bit. The NoC architectures are based on 2D-mesh with XY routing. We ran all QEMUs as user mode emulation, and the ISA of the target cores is i386. Specification of a host computer is described in Table 2.

We use a JPEG encoding application as benchmark program. As shown in Figure 5, JPEG encoding application is pipelined with nine stages, each of which is mapped to a CPU core. The number of JPEG pipelines is depends on the number of cores. When we simulate the NoC architectures with 18 cores, we perform two JPEG pipelines on 18 cores. On the other hand, when we simulate the NoC architectures with 108 cores, we perform 12 JPEG pipelines on 108 cores. The multiple JPEG applications are executed in parallel.

Figure 6 shows the elapsed simulation times for different NoC architectures. X-axis means the number of cores, and Y-axis means the elapsed simulation time. For each number of cores, we also changed the link bandwidth. In Figure 6, red bar labeled “Naxim” means the elapsed simulation time obtained by our previous NoC simulator “Naxim” which supports untimed on-chip interconnect network simulation [7].
Figure 5  Mapping of JPEG applications onto NoC architecture (18 cores, 2 pipelines)

Figure 6  Elapsed simulation times for different NoCs

Figure 6 shows that the simulation time proportionally increases by increasing the number of cores. On the other hand, the simulation time decreases by increasing the link bandwidth because wide link bandwidth decreases the number of flits. This means the wide link bandwidth decreases the traffic in the Noxim. When we compare our NoC simulator with previous simulator “Naxim,” Naxim is more than 10 times faster than our NoC simulator. However, Naxim only supports untimed on-chip interconnection network simulation. Our NoC simulator successfully simulates a 108 core NoC in a practical time, and supports cycle accurate network simulation with acceptable overhead.

V. CONCLUSIONS

In this work, we have developed a hardware/software cosimulator for NoC architectures, which takes advantage of QEMU and Noxim. QEMU simulates each CPU core, and Noxim simulates on-chip interconnection network including routers. Our simulator efficiently runs a multi-core host computer, and simulates cycle accurate on-chip interconnection network with QEMU. Experimental results show the effectiveness of our NoC simulator.

Currently, cycle accurate on-chip interconnection network simulation is available by Noxim. However, the synchronization mechanism between QEMU and Noxim is not established yet. This is our important future work to realize fast and accurate NoC simulator.

ACKNOWLEDGMENT

We would like to thank Mr. Keita Nakajima for his supports to conduct our experiments. This work is supported in part by STARC.

REFERENCES