A Fast Network-on-Chip Simulator with QEMU and SystemC

Keita Nakajima 1 Takuji Hieda 2 Ittetsu Taniguchi 2 Hiroyuki Tomiyama 2 Hiroaki Takada 1

1 Graduate School of Information Science Nagoya University Nagoya, Aichi, Japan
2 Department of Electronic and Computer Engineering Ritsumeikan University Kusatsu, Shiga, Japan

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 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, we have also confirmed the scalability and retargetability of our NoC simulator.

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. Error! Reference source not found. 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 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]. Our NoC simulator consists of 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.

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/applications are executed in parallel. In our NoC simulator, the SystemC simulator and
individual QEMUs are different applications from a viewpoint of host OS. Thus, the SystemC simulator and QEMUs can be executed in parallel on the host computer.

- Our NoC simulator can be executed on multiple host computers since the SystemC simulator 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 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 II describes our NoC simulator. Section III shows a set of experiments and discusses the effectiveness of our simulator. Section IV reviews related works, and Section V concludes this paper with a summary.

II. THE NOC SIMULATOR

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

A. 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. The timing accuracy of our NoC simulator is Approximately Timed (AT) level because of QEMU ¹.

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 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 with XY routing. Our simulator currently supports only store-and-forward switching without virtual channel. In fact, 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.

B. 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.

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. 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. Our NoC simulator takes advantage of this feature of QEMU in order to connect QEMUs to the SystemC simulator. Our NoC 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. This layer is named socket wrapper in Figure 2.

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

¹ Although original QEMU is not cycle-accurate, some research groups such as [3] have studied cycle-accurate simulation based on QEMU. We can utilize their achievements at the cost of degraded portability.
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. A fragment of SystemC code for the top module is presented in Figure 3.

```c
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 3. Part of SystemC code for the top module

### III. EXPERIMENTS

We have conducted a set of experiments to demonstrate the effectiveness of our NoC simulator. Specifically, we have evaluated performance scalability and retargetability of our simulator.

#### A. Experimental Setup

We use a JPEG encoding application as our benchmark program. The JPEG application is pipelined with nine stages, each of which is mapped to a CPU core. We have simulated six NoC architectures with 9, 18, 36, 54, 72, 90 and 108 cores. The NoC architectures are based on 2D-mesh with XY routing. As shown in Figure 4, multiple JPEG applications are executed on the NoC architectures. In case of a 18-core NoC, two images are encoded in parallel. Simulation is performed on two host computers as shown in Table 1.

<table>
<thead>
<tr>
<th>PC1</th>
<th>PC2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core i7 990x @ 3.46GHz (6 cores / 12 threads)</td>
<td>Core i7 970 @ 3.2GHz (6 cores / 12 threads)</td>
</tr>
<tr>
<td>DDR3-1333 12GB</td>
<td>DDR3-1333 12GB</td>
</tr>
<tr>
<td>Ubuntu Linux 32bit</td>
<td>Yellow Dog Linux 64bit</td>
</tr>
</tbody>
</table>

#### B. Scalability

First, we ran our NoC simulator on host PC1. We changed the number of cores on the NoC along with the number of image streams, and measured the simulation time. The ISA of the target cores is MIPS with little endian. The simulation time is shown in blue bars in Figure 5. 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 our NoC simulator.

Next, we ran our simulator on two host computers. SystemC and a half of QEMUs are executed on PC1, and the other half of QEMUs are executed on PC2. The simulation time is shown in red bars in Figure 5. The distributed simulation on the dual host computers is 43% faster than the single-host simulation. 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.

C. 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 use Debian Linux as a target OS. A single-stream JPEG encoder is executed on a nine-core NoC architecture. The host computer is PC1 shown in Table 1. The simulation results are shown in Figure 6. 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.

IV. 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 [6], NNSE [7] and PPNOCS [8]. 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.

Several researchers have developed simulation platforms using QEMU and SystemC such as [9][10][11]. Some of them support simulation of multicore 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 many-core NoCs with 108 cores at a practical speed.

V. CONCLUSIONS

In this work, we have developed a fast simulator of many-core NoC architectures using QEMU and SystemC. QEMU simulates a CPU core, while SystemC is used for simulation of routers and interconnection. Our NoC simulator efficiently runs on a multi-core host computer and even on multiple host computers. Our experiments demonstrate the scalability and retargetability of our simulator.

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

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

REFERENCES