Initial Observations of Hardware/Software Co-Simulation using FPGA in Architecture Research

Taeweon Suh Hsien-Hsin S. Lee
Georgia Institute of Technology
School of Electrical and Computer Engineering
{suhtw,leehs}@ece.gatech.edu

Shih-Lien Lu Johh Shen
Microprocessor Research Lab
Intel Corporation
{shih-lien.l.lu,john.shen}@intel.com

ABSTRACT

This paper demonstrates a new hardware/software co-simulation method that performs execution-driven microarchitecture simulation. Based on an off-the-shelf Pentium-III system that communicates with an FPGA via the Front-Side Bus, all the procedures required to enable such simulation are detailed. Using the platform, we ported a simple memory function from Simplescalar to the FPGA and present our preliminary results and analysis. Reflecting the learnings from our initial observations, we then propose hardware/software co-simulation to accelerate the simulation time of multi-core architecture research for our future work.

1. INTRODUCTION

Currently, the FPGA technology has been advanced enough to model complex chips with the realistic operating frequency. Taking advantage from the current FPGA technology, this paper proposes a hardware/software co-simulation methodology using off-the-shelf system and FPGA as an effort to accelerate the simulation time. A simulator runs on an off-the-shelf machine with one or several parts of the simulator implemented on FPGA. Whenever the simulation reaches a point where the simulator's functions are implemented in the FPGA, the simulator interacts with the FPGA.

The advantage of the conventional software simulation method lies in its flexibility. Architects or designers can easily change the desired simulation parameters to examine the system behavior of different architecture variations. However, its disadvantage is the intolerable simulation time. On the other hand, while a complete hardware emulation can provide significant speedup in examining the system behavior, the flexibility will be nonetheless compromised. In this paper, we attempt to leverage the merits of the software simulation and hardware emulation to retain both the flexibility and performance. We demonstrate the preliminary results of the Simplescalar co-simulation with FPGA and discuss the pros and cons of this approach. Even though the preliminary results emphasize a negative impact of using bus as communication medium in hardware/software co-simulation, we find an opportunity of benefiting from it in multi-core research. This paper concludes with our future plan of using this approach in multi-core research with an Intel internal simulator called SoftSDV.

2. RELATED WORK

FAST [2] was proposed to accelerate cycle-accurate simulation by implementing most of the timing model and key parallelizable parts of the functional model in FPGA. FAST relies heavily on modular, reusable components to construct the timing models, making it easy for users to mix and match components to quickly build new, accurate simulators. Such a simulation environment is expected to boost the simulation performance of at least two or three orders of magnitude. Unlike FAST, our approach focuses on the memory subsystem modeling in FPGA.

Active Cache Emulator (ACE) [4] is an FPGA-based L3 cache emulator developed by Intel Corporation. ACE emulates the L3 cache on a Pentium® III [1] (referred to as P-III hereafter) based server system, where the P-III processor contains an L1 and an L2 cache. Sitting on the Front-Side Bus (FSB), ACE keeps track of the memory transactions and stores appropriate TAG information from address bus, according to the emulated cache size. If a memory transaction misses the L3 TAG stored in the FPGA, a default L3 miss latency is inserted onto FSB using the snoop stall protocol. If a hit is detected, zero or a default hit latency is inserted in the same way. ACE enables the L3 cache modeling and its behavior analysis natively on the off-the-shelf machine for commercial workloads. We use the same ACE board for the different experiment purpose.

Even though ACE provides several advantages over a conventional architecture simulation method, there are two shortcomings due to the limitations of the FSB protocol and physical system configuration. First, there is a limit of the number of processors emulated. ACE is able to emulate L3 cache behavior only for the processors in a system since it runs natively on an off-the-shelf system. For example, if there is two processors on a system with the ACE board, ACE reports L3 cache behavior for those two processors. However, in the pure software simulation, the number of processors can be set as a variable. Therefore, there is virtually no limit to the number of processors simulated. The second shortcoming of ACE is that it is not able to emulate complete non-blocking L3 cache due to the FSB protocol. The FSB protocol does not allow out-of-order completion unless the bus transaction is deferred. The deferred transaction usually applies for the I/O transactions on FSB.

3. METHODOLOGY

Figure 1(a) shows our experiment equipment based on an Intel server system. The original system featured two P-III processors. For our work, we replaced one processor with the FPGA board as shown in Figure 1(b). The P-III runs at 500MHz and FSB operates at 66MHz. As such, the FPGA

[1]Pentium® is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries

board, which sits on FSB, also runs at 66MHz. The system is also equipped with 2GB SDRAM as the main memory. Redhat Linux (kernel version 2.4.8-20) running on the P-III manages the whole system.

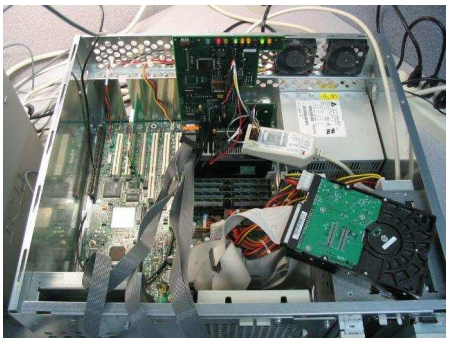
For hardware/software co-simulation, there should be a communication mechanism between P-III and the FPGA board. Since an off-the-shelf system is used in the experiment, there is no special channel for communication other than the FSB and the main memory. Since the FPGA board is in the same level as P-III, there are two communication mechanisms between them. The first one is through the main memory using a lock mechanism. By reserving shared main memory space for communication, one device updates the shared memory after acquiring the lock. Then, the other device reads it later on after the lock acquisition. This approach is costly in terms of execution time since it requires many main memory transactions for each communication. The second mechanism is to use the FSB bus directly. In this approach, the data read operation from the FPGA is somewhat tricky. It should go through the cache-to-cache transfer based on the cache coherence protocol of the P6 FSB. For the cache-to-cache transfer, the FPGA monitors all FSB transactions. If the address of an FSB data transaction is within one of the pre-defined physical address ranges, the FPGA responds to the transaction and provides data to the P-III. We chose the second approach as a communication mechanism since it is much faster than the first approach. Section 3.1 details a Linux device driver to allocate memory pages for the FPGA access and Section 3.2 introduces the cache-to-cache transfer mechanism on the P-III FSB.

3.1 Linux driver

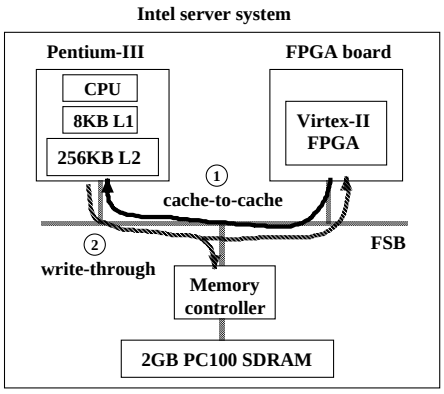
User-level programs under virtual memory support requires address translation that maps a virtual address into a physical address for memory accesses. A conventional processor employs a translation lookaside buffer (TLB) to speed up the address translation. The performance impact of TLBs is modeled in architecture simulators such as Simplescalar [1]. The actual address mapping is managed by the operating system (in our case, the Linux), and the translated physical address, which appears on FSB, is not visible to the users. However, for the hardware/software co-simulation purpose, FPGA needs to know the translated physical address range in advance, with which it decides whether to respond to the FSB transaction or not. Therefore, a special Linux device driver is needed to allocate some of the memory pages for the FPGA access.

From the P-III point of view, the data communication with FPGA means that the P-III should be able to read from FPGA and write to FPGA.

- Read operations can be achieved via the cache-to-cache transfer, through which the FPGA provides data to the P-III. It is illustrated in ① of Figure 1(b). After reading data from FPGA, the P-III must invalidate the cache line where the read data is located. Otherwise, the next read operation to the same location most likely will hit the P-III on-chip cache instead of generating an intended FSB transaction to communicating with the FPGA.
- For write operations, the memory pages in the device driver are configured as write-through mode. Thus, every write operation of P-III to the allocated pages appears on FSB as depicted in ② of Figure 1(b), so the FPGA is able to take data from FSB.



(a) Equipment picture



(b) Equipment schematic

Figure 1: Hardware/software Co-design Equipment

3.2 Cache-to-cache transfer on P-III FSB

The P-III implements a MESI coherence protocol, and the P-III FSB is a 7-stage pipeline bus, consisting of request1, request2, error1, error2, snoop, response, and data. The snoop phase is the 5^{th} pipeline stage, where the snoop results are driven from the remote processors. The cache-to-cache transfer occurs when the FSB transaction hits on an M state line of a remote processor's cache. When a snoop hit occurs on an M-state line, the remote processor asserts HITM# FSB signal. Then, when the memory controller is ready to accept data, which is informed by asserting DRDY# FSB signal, the cache-to-cache transfer takes place. Note that the main memory should be updated simultaneously in the P-III cache coherence protocol when the cache-to-cache transfer occurs. Each cache-to-cache transfer requires to send 4 quadwords data, which are 4 64-bit data transfer.

4. CO-SIMULATION RESULTS

As a preliminary experiment to check the correctness of all procedures, one simple function mem_access_latency() of the Simplescalar was implemented in FPGA and the Simplescalar was modified to generate an FSB transaction whenever it reaches the function in order to get appropriate data from the FPGA. In the Simplescalar, mem_access_latency() is used to model the main memory latency depending on the requested data size. The modified Simplescalar code uses a simple load instruction to get the latency information from the FPGA. The lower 12-bit of the address bus is used to encode the size of the requested data and the upper part of the address bus is loaded with the virtual address of the al-

Table 1: Execution times of baseline and co-simulation

	Baseline (h:m:s)	Co-simulation (h:m:s)	Difference from baseline
mcf	2:18:38	2:20:50	+ 2:12
bzip2	3:03:58	3:06:50	+ 2:52
crafty	2:56:38	2:59:28	+ 2:50
eon-cook	2:43:52	2:45:45	+ 1:53
gcc-166	3:45:30	3:48:56	+ 3:26
gzip_graphic	3:06:51	3:09:05	+ 2:14
parser	3:34:57	3:37:27	+ 2:30
perl	2:42:30	2:45:50	+ 3:20
twolf	2:43:30	2:45:28	+ 1:58

located memory page. Since the TLB translation is on per memory page basis, the lower 12-bit in a 4KB page of the virtual address is identical to that of the physical address. On FSB, the address information is driven in the first stage (the request1 phase) and data is driven in the last stage of the pipeline. Thus, there are a few bus cycles (more than 4 bus cycles) for the FPGA to calculate the appropriate response.

4.1 Simplescalar results

Table 1 shows the co-simulation results using SPECint2000 benchmarks. Each benchmark program was simulated for 1 billion instructions using *sim-outorder*. *Baseline* represents the system without the FPGA implementation. Contrary to the belief, the co-simulation exacerbated the simulation times. On average, the co-simulation is 2 mins 35 secs slower than the baseline. This comes from three factors: the long latency of accessing the FPGA, linux device driver overhead, and the implementation choice of a way too simple Simplescalar function. The following details the each of the factors.

Every Simplescalar's request for the memory latency information goes through FSB as explained. This data read access on FSB takes around 20 FSB bus cycles, which is, in turn, translated to ∼160 CPU cycles. In addition to the native long latency of accessing the bus, there are two contributors of further worsening the latency.

- The cache-to-cache transfer requires to send 4 quadwords data even though the Simplescalar demands only one data (32-bit)
- P-III's coherence protocol requires to update the main memory simultaneously when the cache-to-cache transfer occurs. It means that even when a snoop processor (FPGA in our experiment) is ready to supply data, it has to wait until the memory controller is ready to accept data

Secondly, the device driver takes system resources. For example, one TLB entry is allocated in our device driver for communication between P-III and FPGA. These resources would be used in the software simulation. The other reason of our longer execution times is that mem_access_latency() function is so simple, containing only 21 x86 instructions that even software simulation takes at most a few dozen of CPU cycles, which is translated to only a few FSB cycles. Even though the FPGA can calculate the latency within one bus cycle, it does not show any noticeable advantage over the conventional software simulation.

4.2 Discussion

The Simplescalar co-simulation shows the possibility of using FPGA in architecture research to enhance the simulation time. However, as indicated in the co-simulation results, there are obstacles to overcome before we can make it useful. The FSB access latency is noticeably long in our experimental infrastructure. In addition, there is a system overhead incurred by the linux driver. The driver takes system resources and occupies one TLB entry for FPGA accesses. Therefore, the FPGA will be only helpful when we implement those software functions that need very long execution times in the software simulation. In general, Digital Signal Processing (DSP) functions such as Fast Fourier Transfer (FFT) and Inverse Discrete Cosine Transfer (IDCT) take significant amount of time in software. Thus, those functions are usually implemented in hardware in the embedded system design. However, the computer architecture simulator tends to be control-intensive rather than data-intensive. Therefore, it is not so easy to find a function that would take advantage of hardware implementation unless the whole simulator is implemented in hardware.

Nevertheless, as the mainstream of the future computer architecture research migrates to the multi-core architecture, we find an opportunity of using FPGA to expedite the simulation time. The multi-core simulation takes much more time compared to the single core simulation. Especially, the memory subsystem is getting complicated since distributed L3 caches are likely to be used to reduce L3 access latency, and interconnection network instead of the shared bus is considered to solve the bandwidth problem in a multi-core architecture. Therefore, pure software-based simulation would hinder to characterize architectural variations. Using FPGA as a prototype platform of multi-core architecture was already presented in [3]. In a single Virtex-II Pro FPGA, it implemented up to 5 cores with Xilinx' software core named Microblaze and PowerPC hardcore. Even though this work uses simple RISC processors, it demonstrated the FPGA' potential as an alternative in multi-core architecture research. Section 5 details our plan of the multi-core research using FPGA.

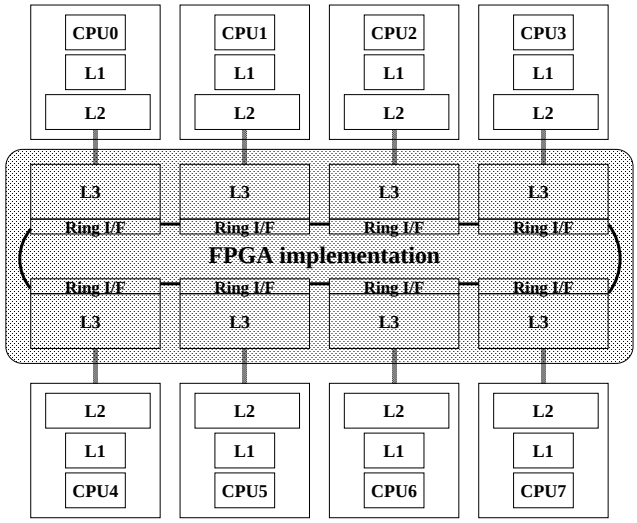
5. FUTURE WORK

Figure 2 shows the simplified diagram of our future work. We begin with 8 core configuration as illustrated in Figure 2(a). The shaded region including L3 caches and interconnection network is implemented in FPGA and the rest is simulated in P-III as shown in Figure 2(b). For L3 caches, FPGA only keeps track of TAG information. As an interconnection network, we first consider a ring architecture.

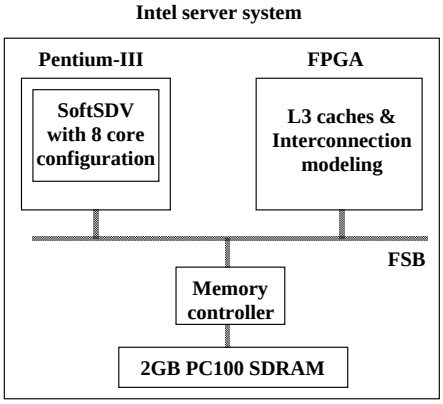
SoftSDV [5] is an Intel internal simulator for platform and microarchitecture research. It can be configured with not only single processor but also multiprocessors, and users can set various parameters such as main memory size and I/O configuration. SoftSDV is able to run Windows or Linux with configured parameters on Windows or Linux-based machines providing real-life environment. SoftSDV co-simulation includes the following details.

- Investigating the cache interface in SoftSDV
- Modifying SoftSDV L3 cache model to access FPGA
- Implementing L3 and ring interconnection in SoftSDV

SoftSDV co-simulation depicted in Figure 2(b) requires at least two FSB accesses for each local L3 miss. Since FPGA



(a) Ring-based multi-core schematic



(b) Partition for multi-core co-simulation

Figure 2: SoftSDV multi-core co-simulation

keeps track of all L3 TAG information, the data access latency can be calculated in the first FSB access, depending on where the data is located (either remote L3(s) or main memory). The second FSB access is used to confirm if the latency information returned from the first access is correct at the simulation time when the calculated access latency is elapsed. This second access is needed since there could be routing problem such as the buffer full in the ring interface by the end of L3 access latency, due to other processors' accesses of the interconnection routes. In addition to the ring configuration issue, there could be many design options depending on L3 cache organization. We expect that the distributed L3 caches and interconnection networks such as torus, ring and mesh in multi-core architecture are complicated enough to implement in hardware to reduce the simulation time.

We also plan to use Pentium® 4 based server system, which accommodates four processors, in our future experiment. With the ACE board in the system, it enables even faster emulation experiment since FSB operates at 100MHz in the system. It could also increase the access frequency to FPGA by multithreading the simulation across the processors, through which co-simulation would benefit more from hardware implementation. In addition, Pentium® 4 based server system creates potential to experiment the behavior of various low-level caches such as the ACE experiment in the P-III system and bus-level research such as the coherence traffic efficiency evaluation in the real multiprocessor environment.

6. REFERENCES

[1] D. Burger and T. Austin. The simplescalar toolset, version 2.0. Technical Report 1342, University of Wisconsin, June 1997.

[2] D. Chiou. FAST:FPGA-based Acceleration of Simulator Timing Models. In *1st Workshop on Architecture Research using FPGA Platforms*, Feb. 2005.

[3] C. R. Clark, R. Nathuji, and H.-H. S. Lee. Using an FPGA as a Prototyping Platform for Multi-core Processor Applications. In *1st Workshop on Architecture Research using FPGA Platforms*, Feb. 2005.

[4] J. Hong, E. Nurvitadhi, and S. L. Lu. Design, Implementation, and Verification of Active Cache Emulator (ACE). To appear in the International Symposium on Field Programmable Gate Arrays (FPGA), Feb. 2006.

[5] R. Uhlig, R. Fishtein, O. Gershon, I. Hirsh, and H. Wang. A presilicon software development environment for the IA-64 architecture. *Intel Technology Journal*, Dec. 1999.