2017 International Conference on High Performance Computing & Simulation

Microbenchmarks for Detailed Validation and Tuning of Hardware Simulators

Rommel Sánchez Verdejo
Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC)
rommel.sanchez@bsc.es

Petar Radojković
Barcelona Supercomputing Center (BSC)
petar.radojkovic@bsc.es

Doctoral Dissertation Colloquium Extended Abstract

I. Introduction and Background

Hardware simulators are indispensable tools for the computer architecture research. They are used by the academia and industry to prototype, explore and evaluate novel microarchitectural features.

The development of hardware simulators is a challenging task as they should mimic the behavior of actual servers comprising complex components such as CPUs, memory and interconnect. The second problem is the implementation details of actual systems are frequently kept as corporate secrets, leaving room for doubt whether the hardware simulators used in the public research are good representatives of the actual systems. Finally, every year new CPU families are released but the computer architecture research is usually based on the CPU simulators that are five to ten years old.

Given the importance of the hardware simulators and the challenges in their development, it is important to pay special attention to their validation. Unfortunately, this process is not standardized. We found two problems with the state-of-the-art simulators validation. First, the metric used for this purpose is overall performance execution of benchmarks suites such as the SPEC CPU 2006 [1]. Second, the complexity of this benchmark suites. More advanced validation may also compare other aspects of the benchmark behavior such as cache misses or branch prediction hit and miss rate.

Our study provides first steps in a systematic methodology to validate hardware simulators. The main objective of the proposed methodology is to compare not only the performance but also the the behavior of the simulator and the actual system. In order to reach this objective, we define a set of resource-stressing microbenchmarks. Each microbenchmark is designed to put a high stress to a single CPU resource, while using the least as possible all other resources. By comparing the execution of the microbenchmarks on the simulator and on the actual system, we can check whether a simulator reproduces the actual system behavior of that specific resource. Since only one resource is stressed at a time, finding the cause of the simulation error is much easier with respect to the case when the validation is performed with complex benchmarks.

In this paper we describe a set of microbenchmarks for the validation of the CPU execution units and the memory subsystem, including on-chip caches and main memory. Also, we present a case study in which the microbenchmarks are used to validate a simulation infrastructure based on the ZSim [2] and DRAMSim2 [3] vs. an actual SandyBridge server. The presented case study shows how the microbenchmarks can be used to isolate the resource behavior of the target architecture and pinpoint the specific differences between the simulator and the target hardware.

II. Our Proposal: microbenchmarks

The first step when tuning a hardware simulator to a given system would be to check the manufacturers' documentation about the target architecture. However, they usually provide incomplete information. For instance, the caches are usually well documented but the main memory access latency is generally avoided. In order to surpass this problem, we propose using RSM to disclose isolated resource behavior of the target system, and for the simulator tuning and validation.

A. Implementation.

The core of the microbenchmarks is implemented directly in assembly of the target processor in order to: (1) Provide the programmer with the maximum control over the instructions that are to be executed; (2) Prevent a compiler from applying any optimization that changes the core of the benchmarks. The assembly functions are inlined in a `C` code in order to avoid the overhead of the function call.

B. CPU Execution units.

All the microbenchmarks are designed using the principle that is presented in Table I. Each benchmark consist of four parts: (1) The register used as a loop iteration counter (`ecx`) is set for 10,000 times of execution; (2) The main section of the benchmark is a sequence of repetitive single instruction of the target ISA; (3) The sequence of target instructions is followed with the decrement of the loop counter register; (4) Finally, the counter value is compared with zero followed by the conditional branch to the beginning of the loop. More than 99.9% of the instructions targets the specific resource we

978-1-5386-3250-5/17 $31.00 © 2017 IEEE
DOI 10.1109/HPCS.2017.135

Line	Source code	Explanation
00001	`mov $10000, %ecx`	Initialize loop counter *ecx* to 10,000
00002	`start_loop:`	beginning of the loop
00003	`ADC %eax, %ebx`	target instruction
00004	`ADC %eax, %ebx`	target instruction
...	`...`	...
10002	`ADC %eax, %ebx`	target instruction
10003	`dec %ecx`	decrement loop counter
10004	`jnz start_loop`	if (counter ≠ 0) jump to start_loop

TABLE I
STRUCTURE OF MICROBENCHMARK ASM CORE.

want to stress. Overhead of looping instructions is negligible. The main loop extends up to 10,000 instructions in order to prevent that the code fits in the *μop cache* of the SandyBridge architecture used in our case study.

C. Caches and main memory

The benchmarks that stress the caches and memory are implemented using the concept of pointer chasing. In the benchmark prologue, we allocate a contiguous section of memory and initialize it to a given array element that contains the address of the next element to fetch. The benchmarks are initialized to (1) Traverse the whole array; (2) Access different cache lines in each memory access; (3) Memory accesses have a random pattern, preventing data prefetchers to bring data to any level of cache. An example of a benchmark memory access pattern is shown in Figure 1.

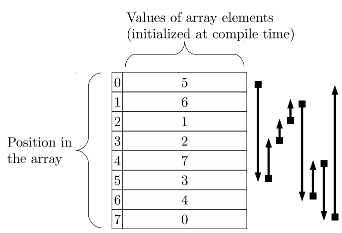

Fig. 1. Example of pointer chasing memory access pattern.

Table II shows the code for memory latency microbenchmarks, which behavior are explain as follows (1) The register used as a loop iteration counter (`ecx`) is initialized; (2) The initial address of the array is passed to the assembly code as an input parameter; (2) The main part of the benchmark is a sequence of indirect load instructions (`mov(%rax), %rax`) that traverse the memory access pattern; (3) The sequence of target instructions is finalized with the decrement of the loop counter register and an exit condition or jump to the beginning of the iteration. The assembly loop is wrapped-up by the `C` program which reads a previously generated file containing information about the array size and the random access pattern.

III. CASE STUDY: ZSIM AND DRAMSIM2 VS. INTEL XEON E5-2670 SANDYBRIDGE-EP

We performed a case study in which the microbenchmarks are used to compare the simulation infrastructure integrated by ZSim and DRAMSim2 simulators. ZSim is a user-level, execution-driven CPU simulator widely used in the computer architecture research, developed to mimic the Westmere architecture and validated against the actual hardware using the SPEC CPU 2006 benchmark suite. It can be easily integrated with the memory simulators such as DRAMSim2 or NVMain [4]. DRAMSim2 is a cycle accurate model of a DRAM memory controller, DIMMs, and buses by which they communicate. It is validated against DRAM manufacturer's Verilog models.

Line	Source code	Explanation
0001	`register struct line *next asm("rax");`	strcut line owns pointer to the next accesss
0002	`register int i asm("ecx");`	`ecx` is the loop counter
0003	`i = 1000000;`	`C` initialization of the loop counter
0004	`next = ptr->next;`	First memory access in `C` form
0005	`start_loop:`	beginning of the loop
0006	`mov (%rax), %rax`	load instruction (pointer chasing)
0007	`mov (%rax), %rax`	load instruction (pointer chasing)
...	`...`	...
1007	`mov (%rax), %rax`	load instruction (pointer chasing)
1008	`dec %ecx`	decrement loop counter
1009	`jnz start_loop`	if (counter ≠ 0) jump to start_loop

TABLE II
STRUCTURE OF MEMORY LATENCY MICROBENCHMARK.

We validate the simulators vs. a dual socket platform. Each socket with an Intel Xeon E5-2670 SandyBridge-EP processor [5] operating at 3.0 GHz. The main memory is 16 GB and is connected to the processors using four DDR3-1600 channels. Each processor runs eight cores, the hyper-threading feature has been disabled like in most HPC systems [6].

A. CPU Execution units

We automate the creation of 346 microbenchmarks covering a wide range of logic for integer and floating point instructions. Then, we executed the benchmarks in actual hardware and the simulator. Results are shown in Figure 2. The horizontal axis of the figure 2 plots an assigned indexed number for each one of the microbenchmarks. The vertical axis shows the relative CPI simulation error calculated as $(CPI_{ZSim} - CPI_{Actual})/CPI_{Actual}$. The results are sorted from the lowest (negative) to the highest (positive) CPI relative error. The red line shows the results for the original ZSim simulator. Out of 346 tested instructions, the simulator matches around 55% the actual system latency while in 28% the simulation error is moderate or high meaning that exceeds 50% of the CPI. Since each of the microbenchmarks stresses one resource at a time, finding the sources of the simulation error is fairly simple. The enhanced version of the simulator show much better accuracy 74% of all the instructions perfectly match the actual system while less than 9% of them show a simulation error of above 50% of the CPI. Moderate or high simulation error come mainly from the complex instructions, or the instructions with CPI dependent on the operand values. Detailed analysis of these cases is an ongoing work.

The case study shows how the microbenchmarks can be used to isolate resource behavior of the target architecture and point the differences between the simulator and the target hardware.

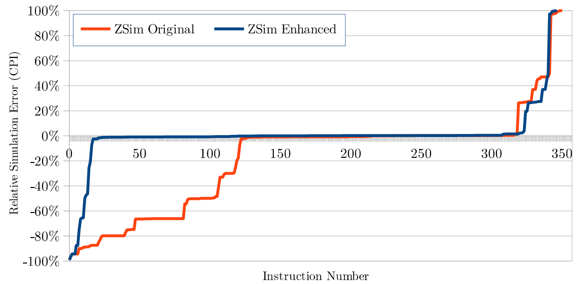

Fig. 2. Relative CPI error comparison between ZSim versions.

B. Caches and main memory

In order to compare the caches and main memory access time between the selected simulators and the actual hardware, we used the same strategy described in Section II-C. The level on the memory hierarchy that will be tested by the microbenchmarks depends on the array size. In our case study, we selected various array sizes in order to stress L1, L2, L3 cache and main memory. Table III abridge this information. ZSim is a user-level simulator, it does not take into account virtual-to-physical address translation and all its overheads such as TLB misses, page walk, cache collision between application data and address-mapping table. To mitigate the address translation overheads in the real system, we used huge memory pages (1 GB per page in our study). For memory latency microbenchmarks we use contiguous memory space. When this space is allocated into huge memory pages, it occupies only few memory pages, and all the memory mapping table entries fit into the TLB. We also quantified the address translation overheads when standard memory pages (4 kB) are used, but this analysis exceeds the scope of this paper.

Comparison between cache and main memory latencies between the actual system and the ZSim+DRAMSim2 simulators is summarized in Figure 3. The horizontal axis of the figure represents the size of the traversed array, while the vertical axis shows the memory access latency in CPU cycles. The figure shows the results for the actual E5-2670 SandyBridge-EP server (yellow line) and the simulators (blue line). From the figure, we can distinguish four steps of the latency corresponding to the L1, L2, L3 cache and main memory. For the L1, L2 and L3 cache the lines overlap, Zsim cache contention based on the manufacturer documentation accurately represents the actual system. However, for the main memory accesses, we detect a significant gap between the simulators and the actual system.

This results motivated us to further explore the sources of this error. Simple integration of ZSim and DRAMSim2 may lead to an underestimation of the main memory access latency. ZSim simulates memory access up to the last level of cache, while DRAMSim2 is focused on the detailed timing simulation of the memory device. This implies that a direct merge of ZSim and DRAMSim2 does not consider the delay contributed by all the circuitry between the last level cache and main memory device, including the memory controller and the memory channel. In order to account for this delay, we introduced the extra latency of 96 cycles between the ZSim and DRAMSim2 that mitigates this gap.

Memory Level, Size and Scope	Number of Measurements	Array sizes (range, stride)
L1 cache 32 kB, Private	8	4 kB to 32 kB , 4 kB
L2 cache 256 kB, Private	16	46 kB to 256 kB, 14 kB
L3 cache 20 MB, Shared	32	888 kB to 20 MB, 632 kB
Main memory, 16 GB/socket	64	83.69 MB to 4 GB, 63.7 MB

TABLE III
SIZE AND ORGANIZATION OF THE CACHES AND MAIN MEMORY OF THE E5-2670 SANDYBRIDGE-EP USED IN THE STUDY.

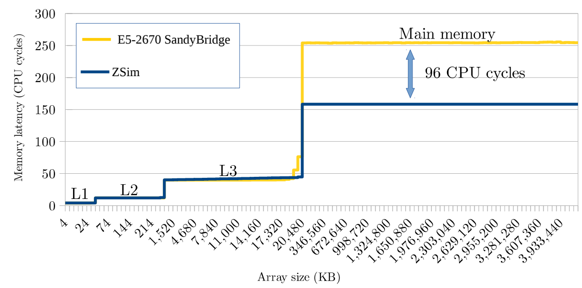

Fig. 3. Memory access latency: L1, L2, L3 cache and main memory.

The memory latency experiments demonstrate that the microbenchmarks can be resourceful to detect specific errors in the simulation configurations that might be overlooked. To this date, there exist no guidelines provided by CPU simulator developers that emphasize the importance of the correct integration with the memory simulators while considering latency of the memory controller and the memory channel.

IV. Conclusions

Our study provides first steps in a systematic methodology to validate computer architecture simulators. We define a set of specific resource-stressing microbenchmarks designed to put a high stress on a single CPU resource. By comparing the execution of the microbenchmark on both systems, we can check whether a simulator indeed reproduces the system behavior for that specific resource. We presented a case study in which the microbenchmarks are used to validate a simulation infrastructure based on the ZSim and DRAMSim2 simulators vs. a real SandyBridge server. The presented case study shows that microbenchmarks can be used to pin-point specific differences between the simulators and the targeted hardware. Overall, our study open a discussion about the validation of the simulators used in the computer architecture community leading to a better and more accurate hardware simulators in the future.

References

[1] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, September 2006.
[2] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *ISCA*, June 2013.
[3] P. Rosenfeld *et al.*, "Dramsim2: A cycle accurate memory system simulator," *IEEE CAL*, 2011.
[4] M. Poremba *et al.*, "Nvmain 2.0: A user-friendly memory simulator to model (non-)volatile memory systems," *IEEE CAL*, 2015.
[5] Intel product specification site. https://ark.intel.com/
[6] Top 500 supercomputer sites. https://www.top500.org/