Validation of the *gem5* Simulator for x86 Architectures

Ayaz Akram[1]
University of California,
Davis, CA
yazakram@ucdavis.edu

Lina Sawalha[2]
Western Michigan University,
Kalamazoo, MI
lina.sawalha@wmich.edu

***Abstract*—*gem5* has been extensively used in computer architecture simulations and in the evaluation of new architectures for HPC (high performance computing) systems. Previous work has validated *gem5* against ARM platforms. However, *gem5* still shows high inaccuracy when modeling x86 based processors. In this work, we focus on the simulation of a single node high performance system and study the sources of inaccuracies of *gem5*. Then we validate *gem5* simulator against an Intel processor, Core-i7 (Haswell microarchitecture). We configured *gem5* as close as possible to match Core-i7 Haswell microarchitecture configurations and made changes to the simulator to add some features, modified existing code, and tuned built-in configurations. As a result, we validated the simulator by fixing many sources of errors to match real hardware results with less than 6% mean error rate for different control, memory, dependency and execution microbenchmarks.**

I. Introduction

Architectural and microarchitectural simulators play a vital role in computer architecture research and performance evaluation as building real systems is very time consuming and expensive. Unfortunately, simulators are susceptible to modeling, abstraction and specification errors [1]. Validation of simulators is important to find out the sources of experimental errors and fixing them, which leads to an increased confidence in simulation results. Although some may argue that when evaluating a new design over a base design, relative accuracy is more important; however, not validating the absolute accuracy of simulators can lead to skewed results and misleading conclusions. Absolute experimental error (compared to real hardware) has been the main method of measuring simulator's inaccuracy and the need for their validation [2]. In this paper, we validate one of the most used computer architecture and microarchitecture simulators, *gem5* [3], against one of Intel's high-performance processors (Core i7-4770). Today's high performance computing (HPC) systems are very heterogeneous; for instance, they can be composed of CPUs of various types, can integrate accelerators and use heterogeneous memory technologies. This makes *gem5* one of the most appropriate architectural simulators to study HPC system architectures, as it allows for simulating heterogeneous systems composed of many cores and complex configurations. *gem5* has been used by ARM research to perform HPC platform simulation and by AMD for their Exascale project [4].

gem5 [3] supports several ISAs (instruction set architectures) like x86, ARM, Alpha, MIPS, RISC-V and SPARC. It supports simple and quick functional simulation and detailed simulation modes. The detailed simulation can be either with system call emulation or with full-system support, in which *gem5* is capable of booting a Linux operating system. *gem5* supports two different pipeline implementations, in-order (IO) and out-of-order (OoO). *gem5* has been previously validated for ARM based targets [5], [6]. However, there does not exist any validation effort for x86 based targets (which is the primary ISA for most of the HPC systems today). In this work, we evaluate the accuracy of *gem5* by comparing its results to those of real hardware (Haswell microarchitecture). Then we point out the sources of errors in this simulator and discuss how we resolved those errors to achieve higher accuracy. Finally, we validate the simulator for an out-of-order core (OoO) of Intel Core i7-4770. Our results show a significant reduction in the absolute mean error rate from 136% to less than 6% after validation.

The paper is organized as follows: Section II describes our validation methodology for *gem5* simulator. Section III describes the sources of inaccuracies found in *gem5*, and discusses the modifications that we performed to validate the simulator. Section IV discusses related work and section V, concludes the paper.

II. Validation Methodology

In many HPC systems, performance evaluation or simulation validation of different components can be done using a mathematical model, for example, network simulation. However, building a mathematical model is not suitable for validating microarchitecture simulators because of the huge amount of configurations and details in such simulators, specifically *gem5*. In addition, many simulation results are not known without knowing the runtime behavior of programs. Thus, building a mathematical model will not result in an accurate validation of such simulators.

[1]Ayaz Akram finished this work while at Western Michigan University.
[2]Corresponding Author: Lina Sawalha (lina.sawalha@wmich.edu)

978-1-7281-5977-5/19/$31.00 ©2019 IEEE
DOI 10.1109/PMBS49563.2019.00012

In this work, we validate gem5 by comparing its results with the results obtained from real hardware runs using hardware monitoring counters (HMCs). We use *perf* tool to profile the different execution statistics with HMCs [7]. Then we find the sources of inaccuracies in *gem5* using the calculated experimental error and by correlating the inaccuracies in simulated performance to different simulated architectural statistics and pipeline events. We configured *gem5* as close as possible to an Intel Haswell microarchitecture-based machine (Core i7-4770), relying on multiple available sources [8]–[10]. Table I shows the main configurations of the microarchitecture. The single-core experiments, which we focus on in this work, will eventually help to improve the accuracy of multi-core and multi-node HPC system simulations.

TABLE I: Target Configurations.

Parameter	Core i7 Like
Pipeline	Out of Order
Fetch width	6 instructions per cycle
Decode width	4-7 fused μ-ops
Decode queue	56 μ-ops
Rename and issue widths	4 fused μ-ops
Dispatch width	8 μ-ops
Commit width	4 fused μ-ops per cycle
Reservation station	60 entries
Reorder buffer	192 entries
Number of stages	19
L1 data cache	32KB, 8 way
L1 instruction cache	32KB, 8 way
L2 cache size	256KB, 8 way
L3 cache size	8 MB, 16 way
Cache line size	64 Bytes
L1 cache latency	4 cycles
L2 cache latency	12 cycles
L3 cache latency	36 cycles
Integer latency	1 cycle
Floating point latency	5 cycles
Packed latency	5 cycles
Mul/div latency	10 cycles
Branch predictor	Hybrid
Branch misprediction penalty	14 cycles

Our adopted methodology for the validation of *gem5* simulator is based on the use of a set of microbenchmarks to focus on problems with specific sub-systems of the modeled CPU. The microbenchmarks used in this work are the same as the ones used for the validation of SiNUCA microarchitectural simulator [11], which were inspired by the microbenchmarks used for SimpleScalar simulator's validation [2]. The used microbenchmarks are divided into sets of four control, six dependency, five execution, and six memory microbenchmarks. Table II shows a summary of the different microbenchmarks used in this validation.

The first step in our validation methodology is to perform statistical analysis to find a correlation (Pearson's correlation coefficient [12]) of different simulation statistics and the percentage error in the simulated performance statistic (compared to the real hardware). The specific simulation statistics we use, include: number of instructions, micro-operations, load operations, store operations, branch operations, floating point operations, integer operations, branch mispredictions, cache misses, speculated instructions, squashed instructions (after misspeculation happens), execution cycles, and full-queue events for various pipeline stages. The main performance statistic we use in this work is instructions per cycle (IPC). Then we analyze the correlation and simulated performance results to discover the possible sources of error in the simulator. This knowledge is combined with detailed results and analysis of *gem5*'s code to improve the simulation model. Finally, based on our findings, we apply two types of changes in the simulator, configurational changes and code changes. To discuss the configurational calibration, we refer to the base configuration of the simulator as *base_config* and the calibrated configuration as *calib_config*. *base_config* models Haswell on *gem5* based on the parameters listed in Table I, but this configuration is set without an in-depth knowledge of the simulator, as any new user would do. *calib_config* is the configuration, which is achieved based on the initial correlation analysis and knowledge of specific implementation details of the simulator. This calibration technique helps to improve the accuracy of our results. Code changes include both modifying existing code, to fix the errors that we found in the simulator, and developing and adding new features/optimizations to *gem5*, to better represent the Haswell microarchitecture. Most of our changes and tuned configurations are available on github[1].

III. Results and Analysis

A. Observed Inaccuracies

Figure 1 shows the values of IPC (instructions per cycle) for all microbenchmarks on the unmodified and the modified versions of the simulator normalized to the IPC values on real hardware. Figure 2 shows the average percentage inaccuracy observed on both versions of the simulator for all categories of the microbenchmarks separately. As shown in Figure 2, the average inaccuracy for the unmodified simulator can become very large. The observed average error in IPC values is 39% for control benchmarks, 8.5% for dependency benchmarks, 458% for execution benchmarks and 38.72% for memory benchmarks. As discussed in the previous section, in order to specify the possible sources of inaccuracies in the simulation model, we also calculated the correlation between various *gem5* events (statistics produced by the simulator) and percentage error in IPC values produced by the simulator as shown in Figure 3. The figure shows an average of Pearson's correlation coefficient (for all benchmarks), which can have a maximum value of +1 and a minimum value of -1. Positive correlation indicates a positive error in IPC value (overestimation of IPC by the simulator) and a negative correlation indicates a negative error in IPC values (underestimation of IPC by the simulator). Most of the

[1]https://github.com/ayaz91/gem5Valid_Haswell.git

TABLE II: Description of the Used Microbenchmarks.

MicroBenchmarks	Target	Construct
Control Benchmarks		
Control Conditional	conditional branches	if-then-else in a loop
Control Switch	indirect jumps	switch statements in a loop
Control Complex	hard-to-predict branches	if-else and switch statements in a loop
Control Random	random behavior of branches	if-then-else in a loop and branch outcome determined by a shift register
Dependency Benchmarks		
Dep1	dependency chain	dependent instructions with dependency length of 1 instruction
Dep2	dependency chain	dependent instructions with dependency length of 2 instructions
Dep3	dependency chain	dependent instructions with dependency length of 3 instruction
Dep4	dependency chain	dependent instructions with dependency length of 4 instructions
Dep5	dependency chain	dependent instructions with dependency length of 5 instructions
Dep6	dependency chain	dependent instructions with dependency length of 6 instructions
Execution Benchmarks		
Ex int-add	integer adder	32 independent instructions in a loop
Ex int-multiply	integer multiplier	32 independent instructions in a loop
Ex fp-add	floating point adder	32 independent instructions in a loop
Ex fp-multiply	floating point multiplier	32 independent instructions in a loop
Ex fp-division	floating point divider	32 independent instructions in a loop
Memory Benchmarks		
Load Dependent 1 & 2	memory hierarchy for dependent load instructions	linked lists walked in loops
Load Independent 1 & 2	memory hierarchy for independent load instructions	different cache sizes with 32 parallel independent loads in a loop
Store Independent	memory hierarchy for independent store instructions	different cache sizes with 32 parallel independent stores in a loop

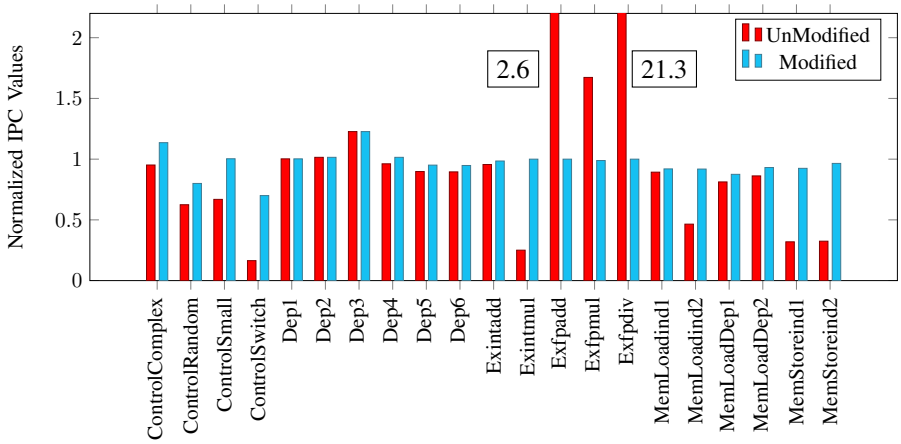

Fig. 1: Normalized IPC values of microbenchmarks compared to the real hardware

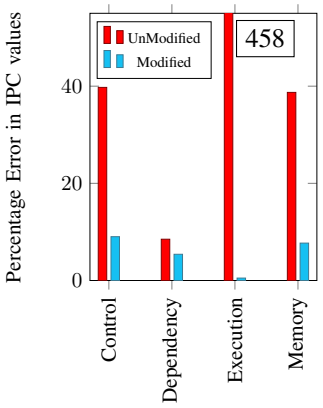

Fig. 2: Average accuracy for microbenchmarks compared to the real hardware

cache related events at all levels (cache misses and accesses) show negative correlation with the percentage error in IPC values, indicating a possible overestimation of cache/memory latencies. Similarly, many of the pipeline stall events (like ROBFullEvents, IewSQFullEvents, IewIQFullEvents in *gem5*) show a strong negative correlation with the percentage error. This indicates that the number of wasted cycles in the simulator in case of pipeline stall events are overestimated, thus underestimating of performance. The number of executed micro-operations (μ-ops) shows a positive correlation with the percentage error in IPC values meaning that the benchmarks with a very high number of μ-ops usually lead to overestimated IPC values. The number of store operations shows a very strong negative correlation explaining the low accuracy of the *memory store_independent* benchmarks shown in Figure 3. This strong negative correlation should be looked in association with strong negative correlation of IewSQFullEvents (store-queue-full events), which shows that the main reason for high inaccuracy in case of *memory store_independent* benchmarks would be an overestimated wasted number of cycles in case of a store-queue-full event. A negative correlation of branch mispredictions with the percentage error in IPC suggests that the branch predictor is a potential source of error. This was also indicated in previous works [13], [14], as actual branch predictors used in the real hardware are not published and branch predictors of *gem5* appear to lag in performance when compared to the actual hardware.

B. Applied Modifications to the Simulator

We use the knowledge obtained from previously discussed statistical analysis to guide tuning and modifications in the simulator. The applied changes to the simulator and the configurational calibration improved the accuracy of the simulator as shown in Figures 1 and 2. For the modified version of

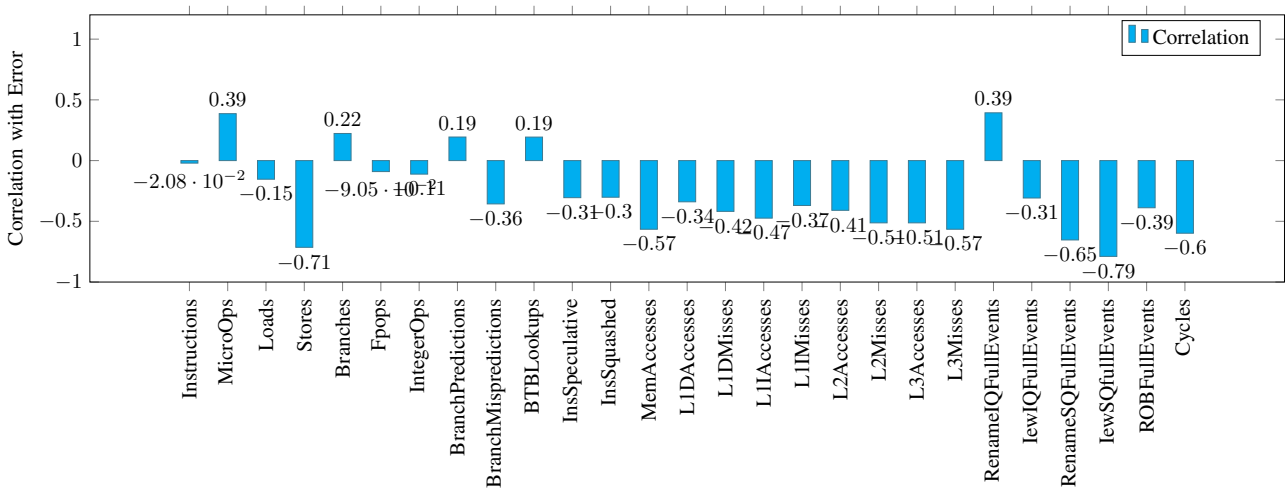

Fig. 3: Correlation of *gem5* events with the percentage error in performance (IPC)

the simulator, the observed average error in IPC values is reduced to 9% for control benchmarks, 5.4% for dependency microbenchmarks, 0.5% for execution microbenchmarks and 7.7% for memory benchmarks. Next, we discuss the changes made to the simulator (or configuration) in the context of observed problems.

We found that increasing the value of *issueToExecute* delay parameter (delay between the issue and the execute stages of the pipeline) beyond one, inhibits "back-to-back" execution of two instructions, (this issue has been observed by others as well [15]). In *base_config* of the simulator, this parameter was set as two along with different delay values between the pipeline stages, so that the overall required pipeline length was achieved. Changing the value of *issueToExecute* delay to one in *calib_config* improves the IPC values for many microbenchmarks (and improves accuracy). For example, this change reduced the inaccuracy in *control_small* microbenchmark from 33.6% to only 0.4%. In *base_config* the instruction cache (i-cache) access latency was set as four cycles (as in the real hardware). We observed that fetch unit in the out-of-order (OoO) pipeline of *gem5* does not request to i-cache while it is waiting on a response from i-cache (as also discussed here: [16]). In other words, even if the i-cache is configured as a non-blocking cache, the fetch unit makes it behave as a blocking cache. This reduces the fetch unit's throughput considerably. One possible solution is to keep the i-cache hit latency equal to one and add an additional latency in the front end stages of the pipeline to compensate for the missing i-cache latency [16]. Specifically, this extra latency is added between the fetch and the decode stages (in *fetchToDecode* delay parameter) in *calib_config*. This change particularly improved the IPC of *control_switch* by a factor of three (and thus improved its accuracy).

We also found that microoperations (μ-ops) to instructions ratio was very high in many cases in *gem5*. Thus, we changed the source code of the simulator to make x86 instructions to μ-ops decoding more practical. To make these changes we used information available in [9] and also looked at the code of other x86 simulators such as ZSim [17] and Sniper [18], which exhibit more realistic values of μ-ops to instructions ratios. Our modifications achieved an x86 (μ-ops) to instructions ratio within 5% of what is observed on the real hardware and other sources [19], [20], for SPEC-CPU2006 benchmarks. Since, in *gem5*, all pipeline widths are defined in terms of μ-ops and all pipeline structures (like reservation stations (RS) and reorder buffers (ROB)) hold μ-ops, reducing the values of μ-ops to instruction ratios improves the overall IPC values and improves the accuracy as well. For example, this change reduced the inaccuracy from 10.5% to 5.1% in *dep5* microbenchmark and from 6.5% to 1.5% in *ex_int_add* microbenchmark. The labelling of different classes of μ-ops in *gem5* needs improvement as well. We noticed that the floating point multiplication and division operations are wrongly classified as floating point add operations in *gem5*. We modified *gem5*'s code to correct their labels, which eliminated the large inaccuracies observed in floating point related microbenchmarks *(ex_fp_mul and ex_fp_div)* shown in Figure 1.

Memory_load_ind2 microbenchmark loads independent data values from L1 data cache (of 32KB size) with an offset of 64 bytes (the cache line size). The accesses occur in a loop and the number of data cache misses should be very low in this case as the memory footprint is the same as the size of L1 data cache. The simulator showed a significant number of data cache misses for this microbenchmark. Digging into this problem, we observed that the origin of the problem is related to the branch prediction. The microbenchmark consists of a nested loop, where the inner loop is responsible for traversing through the data array, while the outer one repeats this process a specific number of times. It is observed that on every last iteration of the inner loop, the branch instruction (responsible for this loop) is mispredicted to be taken. This

TABLE III: Branch Miss Ratio

Benchmark	Real	SimUnMod	SimMod
ControlComplex	0.0012	0.000205	0.00021
ControlRandom	28.26	30	28.78
ControlSmall	0.0004	0.000068	0.00007
ControlSwitch	0.0011	0.01609	0.000179
MemLoadind1	0.0576	0.00936	0.00947
MemLoadind2	0.0239	5.559	0.00513
MemLoadDep1	0.0698	0.0093	0.00953
MemLoadDep2	0.0528	5.55975	5.559
MemStoreind1	0.0528	0.009387	0.00956
MemStoreind2	0.0298	5.56	0.00513

TABLE IV: L1D Cache MPKI

Benchmark	Real	SimUnMod	SimMod
MemLoadind1	0.01065	0.0020794	0.00209926
MemLoadind2	0.114602	120.399	0.00156352
MemLoadDep1	0.073911	0.001996	0.00200671
MemLoadDep2	15.825	25.774	25.774077
MemStoreind1	0.01583	0.001935	0.001996
MemStoreind2	0.132615	0.001522	0.001547

results into access of a new cache block (on wrong path) in set 0 of the cache, kicking out a useful block. This in turn causes a high number of data cache misses leading to lower IPC values. We implemented a loop predictor in *gem5* that eliminated the problem and improved the accuracy of this type of microbenchmarks as shown in Figure 1. Similar is the case with *Memory_Store_ind2* microbenchmark.

We also observed that *gem5* is quite conservative in terms of its pipelining behavior in the front-end stages of the pipeline. Instead of proper queuing in case of a blocking events, *gem5* has SkidBuffers at each stage that absorb inflight instructions whenever there is a blocking event. Moreover, in case of removal of blocking event, any stage will get unblocked completely only after its SkidBuffer is fully drained, and then would allow the previous stages to unblock as well. For example, in a worst case scenario, pipeline bubbles (or stalls) is equal to a sum of delay between decode and rename stages and delay between fetch and decode stages in case of a rename stage block. This behavior has also been observed by [15], [21]. This behavior should be analyzed in association with the observed strong negative correlation of pipeline stall events and the percentage error in simulation results (shown in Figure 3) as discussed previously. To make the pipelining behavior less conservative, we might need more structural changes in the pipeline of the simulator. For the moment, we relied on tweaking of the pipeline configuration (in *calib_config*) to decrease the effect of additional pipeline bubbles. We figured out that we can change the values of the backward path delays between stages of the pipeline (which are set to 1 by default). Since these delays are set to 1, whenever any stage gets blocked it sends a message to previous stages to block as well on the next cycle irrespective of any spaces in queues between the stages and the stages' SkidBuffers (*gem5* structures used to store instructions in case of blocking events). Setting the values of delays on backward paths same as forward path for a particular stage can avoid most of the wasted cycles in case of stalls, as any blocking stage would wait for the number of set cycles before conveying the blocking message to the previous stage. This change improved the performance specifically for $control\_random$ and $control\_switch$ microbenchmarks.

Furthermore, we studied a couple of microarchitectural events to observe the differences in their values on the simulator and the real hardware. Tables III and IV show a comparison of the real hardware, the unmodified simulator and the modified version of the simulator using two microarchitectural events (branch miss ratio (branch misses per thousand branch instructions) and L1 data cache misses per thousand instructions). The tables show only those microbenchmarks for which the events are relevant, control and memory microbenchmarks. Generally the difference between the values of these events in the modified and unmodified versions of the simulator is small. Moreover, the simulated event counts are generally underestimated. These event counts should be analyzed carefully as the inaccuracy observed in most of the cases is insignificant due to the small values of the events, and it does not affect the overall performance results of the simulator. The difference in events' values of the simulator and that of the real hardware can also be attributed to the noise in the measured values of events on the real hardware, which can become visible when the counts of the measured events are very small as is the case for most of the microbenchmarks. Few interesting cases where the accuracy in the values of the events in the modified version of the simulator is improved significantly include: *control_random*, *memory load_ind2* and *memory store_ind2* for branch miss ratios and *memory load_ind2* for L1D cache MPKI (misses per thousand instructions).

IV. Related Work

There are many validation works of different simulators in the literature for different architectural and microarchitectural simulators, for example, SimpleScalar [2], SiNUCA [11], ESESC [22], ZSim [19], MARSSx86 [23], *gem5* (ARM ISA) [5], [6], Multi2Sim [24], Sniper [25], and McSimA+ [26]. Nowatzki et al. [27] presented different observed issues with four common performance and power simulators including *gem5*. The issues they found with *gem5* include wrongly classified and inefficient μ-ops and inconsistent pipeline replay mechanism.

Weaver and Mckee [28] showed that the use of unvalidated cycle-by-cycle simulators with large inaccuracies cannot be justified in comparison to fast tools like emulators, which can provide similar results at a much higher speed. Asri et al. [23] calibrated MARSSx86 to simulate "architecture-rich" heterogeneous processors. They showed that unvalidated simulators can result in significant differences between simulation and real hardware results, which can be more problematic for accelerator studies. Akram and Sawalha [14], [29], [30] compared experimental errors for different architectural simulators including *gem5* and concluded that

the main sources of inaccuracies in simulation results are overestimated branch mispredictions, imprecise decoding of instructions into microoperations and high cache misses. Walker et al. [13] recently presented *GemStone*; a systematic tool to compare reference hardware and simulator results, identify sources of error and apply modifications to the simulation model to improve its accuracy using statistical analysis. They used ARM Cortex-A15 *gem5* model, to present *GemStone*, and identified branch prediction to be the main source of error in simulation results. Our work validates *gem5* against an x86 single-core target platform, which has not been performed before.

V. Conclusion

Validation of architectural simulators is important as it increases the confidence in the results of simulators. While *gem5* has already been validated for ARM based processors, there does not exist any validation effort for x86 based processors to the best of our knowledge. In this work, we have identified various errors in the x86 out-of-order CPU model of *gem5*. The accuracy of x86 model of *gem5* has been improved significantly (reduction in mean error rate from 136% to 6% for the studied microbenchmarks) after applying different modifications in the simulator. Our modifications included both simulator configuration calibration, code modification and adding new structures to the simulator. In the future, we would like to dig deeper into some of the problems that we already identified here to further reduce the inaccuracy and expand our study to use more realistic HPC benchmarks and multi-core/multi-node HPC architectures.

References

[1] B. Black and J. P. Shen, "Calibration of Microprocessor Performance Models," *Computer*, vol. 31, pp. 59–65, May 1998.

[2] R. Desikan, D. Burger, and S. W. Keckler, "Measuring Experimental Error in Microprocessor Simulation," in *Proceedings of the 28th annual International Symposium on Computer Architecture*, pp. 266–277, Gothenburg, Sweden, Gothenburg, Sweden, June 30-July 4 2001.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, May 2011.

[4] M. J. Schulte, M. Ignatowski, G. H. Loh, B. M. Beckmann, W. C. Brantley, S. Gurumurthi, N. Jayasena, I. Paul, S. K. Reinhardt, and G. Rodgers, "Achieving Exascale Capabilities through Heterogeneous Computing," *IEEE Micro*, vol. 35, no. 4, pp. 26–36, 2015.

[5] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy Evaluation of GEM5 Simulator System," in *IEEE 7th International Workshop on Reconfigurable Communication-centric Systems-on-Chip*, pp. 1–7, York, UK, York, UK, 9-11 July 2012.

[6] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver, "Sources of Error in Full-System Simulation," in *IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 13–22, Monterey, CA, 2014.

[7] Perf, "perf: Linux profiling with performance counters," 2015. Retrieved August 5, 2015 from https://perf.wiki.kernel.org/index.php/Main_Page.

[8] "Intel® 64 and IA-32 Architectures Software Developer Manuals." Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html, [Online, accessed Aug., 2019].

[9] A. Fog, "Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs." Available: http://www.agner.org/optimize/instruction_tables.pdf, [Online, accessed 3 September, 2016].

[10] *Intel's Haswell CPU Micrarchitecture*. Available: http://www.realworldtech.com/haswell-cpu/ [Online; accessed 1-June-2017].

[11] M. A. Z. Alves, C. Villavieja, M. Diener, F. B. Moreira, and P. O. A. Navaux, "SiNUCA: A Validated Micro-Architecture Simulator," in *IEEE High Performance Computing and Communications (HPCC)*, pp. 605–610, 2015.

[12] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson Correlation Coefficient," in *Noise reduction in speech processing*, pp. 1–4, Springer, 2009.

[13] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi, "Hardware-validated cpu performance and energy modelling," in *IEEE International Symposium on Performance Analysis of Systems and Software*, (Belfast, UK), pp. 44–53, April 2018.

[14] A. Akram and L. Sawalha, "x86 Computer Architecture Simulators: A Comparative Study," in *IEEE 34th International Conference on Computer Design (ICCD)*, pp. 638–645, 2016.

[15] "CPU configuration (gem5-users)." https://www.mail-archive.com/gem5-users@gem5.org/msg12282.html, 2015. [Online; accessed 5-August-20157].

[16] "O3 fetch throughput when i-cache hit latency is more than 1 cycle (gem5-users)." https://www.mail-archive.com/gem5-users@gem5.org/msg10393.html, 2015. [Online; accessed 5-August-2017].

[17] D. Sanchez and C. Kozyrakis, "ZSim: Fast and Accurate Microarchitectural Simulation of Thousand-Core Systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pp. 475–486, June 2013.

[18] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the Level of Abstraction for Scalable and Accurate Parallel Multi-Core Simulation," in *ACM International Conference for High Performance Computing, Networking, Storage and Analysis*, (Seattle, WA), pp. 1–12, Nov. 2011.

[19] Zsim, *Zsim Tutorial Validation*, 2015. Available: http://zsim.csail.mit.edu/tutorial/slides/validation.pdf [Online; accessed 5-June-2017].

[20] *ISA POWER STRUGGLES*. Available: http://research.cs.wisc.edu/vertical/wiki/index.php/Isa-power-struggles/Isa-power-struggles [Online; accessed 5-June-2017].

[21] T. Tanimoto, T. Ono, and K. Inoue, "Dependence Graph Model for Accurate Critical Path Analysis on Out-of-Order Processors," *Journal of Information Processing*, vol. 25, pp. 983–992, 2017.

[22] E. K. Ardestani and J. Renau, "ESESC: A Fast Multicore Simulator Using Time-Based Sampling," in *IEEE 19th International Symposium on High Performance Computer Architecture*, pp. 448–459, Shenzhen, China, Shenzhen, China, 23-27 February 2013.

[23] M. Asri, A. Pedram, L. K. John, and A. Gerstlauer, "Simulator Calibration for Accelerator-Rich Architecture Studies," in *International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, Samos, Greece, July 2016.

[24] R. Ubal, B. Jang, P. Mistry, D. Schaa, and D. Kaeli, "Multi2Sim: A Simulation Framework for CPU-GPU Computing," in *Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques*, pp. 335–344, Minneapolis, MN, 2012.

[25] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An Evaluation of High-Level Mechanistic Core Models," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 3, Article 28, 2014. 25 pages.

[26] J. H. Ahn, S. Li, O. Seongil, and N. P. Jouppi, "McSimA+: A Manycore Simulator with Application-level+ Simulation and Detailed Microarchitecture Modeling," in *IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 74–85, Austin, Tx, 2013.

[27] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam, "Architectural Simulators Considered Harmful," *IEEE Micro*, vol. 35, no. 6, pp. 4–12, 2015.

[28] V. M. Weaver and S. A. McKee, "Are Cycle Accurate Simulations a Waste of Time," in *7th Workshop on Duplicating, Deconstructing, and Debunking (WDDD)*, pp. 40–53, 2008.

[29] A. Akram and L. Sawalha, "A Comparison of x86 Computer Architecture Simulators," Tech. Rep. TR-CASRL-1-2016, Western Michigan University, MI, October 2016.

[30] A. Akram and L. Sawalha, "A Survey of Computer Architecture Simulation Techniques and Tools," *IEEE Access*, 2019.