Profiling gem5 Simulator

Johnson Umeike, Neel Patel, Alex Manley, Amin Mamandipoor, Heechul Yun, Mohammad Alian
Electrical Engineering and Computer Science Department
University of Kansas

***Abstract*—In this work, we set out to find the answers to the following questions: (1) Where are the bottlenecks in a state-of-the-art architectural simulator? (2) How much faster can architectural simulations run by tuning system configurations? (3) What are the opportunities in accelerating software simulation using hardware accelerators? We choose gem5 as the representative architectural simulator, run several simulations with various configurations, perform a detailed architectural analysis of the gem5 source code on different server platforms, tune both system and architectural settings for running simulations, and discuss the future opportunities in accelerating gem5 as an important application. Our detailed profiling of gem5 reveals that its performance is extremely sensitive to the size of the L1 cache. Our experimental results show that a RISC-V core with 32KB data and instruction cache improves gem5's simulation speed by 31%∼61% compared with a baseline core with 8KB L1 caches. Our paper is the first step toward building specialized hardware and software environments for accelerating software-based simulators.**

I. Introduction

We are in the golden age of computer architecture [1] where the continuation of Moore's law is premised upon the specialization of hardware for different application domains. This simply means that computer architects are going to design many more hardware in the years to come.

Software-based simulation is the backbone of computer architecture research and development. Since the inception of computer architecture as a field, many software-based architectural simulators[1] and simulation techniques have emerged [2]. Currently, various architectural simulators are in use by academia and industry for modeling different aspects of future computing platforms. gem5 [3], Sniper [4], MARSSx86 [5], and ZSim [6] are just a few examples of architectural simulators currently with active communities. Designing hardware requires many hours of simulation and this figure will only increase in the future due to the proliferation of open-source hardware [7] and the need for domain-specific hardware design.

Improving simulation performance has been in the spotlight from the early implementations of software-based simulators. Throughout the years, many techniques such as parallelizing simulation on multiple cores [6] or multiple nodes [8], [9], using hardware virtualization support [10], [11], sampling techniques [12], [13], [14], trading off simulation accuracy for speed [15], [16], and using configurable hardware for modeling flexible systems [17], [18], [19] have been proposed and implemented to improve simulation performance. Previous works often overlook simple software and system optimizations that can significantly improve the simulation speed without introducing complex changes to the simulator. In this work, we set to fill this gap.

gem5 is one of the most widely used architectural simulators, providing a platform for modeling future computer systems [20]. gem5 also supports various modes of execution as well as different levels of simulation detail. Due to the ubiquity of gem5 and its large user base, we select gem5 as the representative simulator for this work. We simulate different workloads on gem5 with diverse configurations, profile gem5 code, and perform a detailed architectural analysis of the gem5 execution to find the bottlenecks in the official gem5 release. We compare the simulation time, (measured as host seconds) when running gem5 on two different platforms: Intel Xeon and Apple M1 Chips. We perform a detailed comparison of the architectural statistics between the platforms. We also run gem5 on FireSim [17], which is an FPGA-accelerated architectural simulator, to investigate the sensitivity of gem5's speed to some architectural parameters. Finally, we use our profiling insights to perform simple system tuning and propose architectural recommendations to improve gem5 simulation speed.

This work is the first step towards better understanding the characteristics of a state-of-the-art architectural simulator and developing hardware and software solutions to meet the growing demand for architectural simulation. Our major contributions in this paper hinge upon answering the following questions:

- **Where are the bottlenecks in running gem5 on a Xeon server?** Our results show that gem5 is extremely front-end (instruction) bound with large iCache and TLB miss rates. Due to the huge code size, an abundance of virtual functions, and runtime polymorphism in the source code, there is no particular hot function or code block in gem5. As a result, the decoder unit in an out-of-order processor is under extreme pressure to supply μOps to the back-end, and there is large misprediction and resteer overhead in the pipeline's front-end.
- **How does the performance of gem5 vary in different server platforms when running simulations?** When running architectural simulations, the focus is usually on the configuration of the simulated system but the configuration of the host is often ignored. Our results show that the underlying physical hardware notably impacts simulation time. For instance, a MacBook Pro with an M1 chip completes the same simulation 1.7×∼3.02× faster than a server equipped with Xeon Gold 6242R CPUs and 96GB of DDR4 DRAM.
- **How much faster can gem5 run by tuning the architectural, system, and runtime configurations on the host?** Motivated by the observations from running

[1]Unless mentioned otherwise, throughout the paper, we refer to "software-based architectural simulators" as "simulators"

gem5 on different server platforms, we study the sensitivity of gem5 speed to L1/L2 cache size, CPU frequency, back gem5 code with huge pages, and recompile gem5 for optimization. Our results show that larger L1 data and instruction caches can significantly speed up gem5 simulations.

- **Long term solution?** Lastly, we discuss some of the solutions moving forward, such as designing specialized accelerators for simulation.

The rest of this work is organized as follows. In Sec.II, we discuss the motivation for this work and provide background information on computer architecture simulators. In Sec.III, we describe our methodology for profiling gem5 and collecting experimental results. In Sec.IV, we analyze gem5 source code and microarchitectural statistics and reveal the runtime execution characteristics of gem5 running on different platforms. In Sec.V we discuss the sensitivity of gem5 speed to varying system and architectural configurations of the simulation server. Sec.VI summarizes some of our takeaways and discusses future directions for accelerating simulation speed. Sec.VIII concludes this work.

II. Motivation and Background

Simulation is extensively used in both academia and industry. Although the bar for the accuracy of simulation in academic research is lower, potentially impactful academic research requires full-system modeling of various hardware and software components [21]. Ideally, we would have a simulator that is as fast as the real hardware, as flexible as a software implementation, and performant as the target hardware. However, the speed of simulation, and complexity of implementation are influenced by the required simulation detail [2].

We can classify architectural simulation into two categories: functional and timing. Functional simulation (i.e., emulation) models the functionality of future hardware. This is mainly used for validating hardware functionalities and software development and testing before the hardware is built. Some examples are HASE[22], simCore [23], Barra [24], Simics [25], AtomicSimple CPU configuration in gem5 [3]. Timing simulation (i.e., performance simulation) is used to model the timing aspects of hardware while providing the correct functionality. There are timing simulators with different performance-modeling fidelity. Clearly, the complexity of a simulator proportionally increases with its modeling fidelity. Some examples are zSim [6], sniper [4], HAsim [18], gem5 [3], Simple Scalar [26]. Additionally, simulators can operate in user-level or full-system mode. In the user-level mode, the simulator only executes user-level code without modeling the operating system. System calls are bypassed and serviced by the underlying host. This mode is also referred to as system-call emulation mode. On the other hand, a full-system simulator models the entire computing system, including memory, and I/O subsystems while running an unmodified operating system.

gem5 is a state-of-the-art architectural simulator with an active user and developer community. It is extremely configurable, supports multiple ISAs, and can perform full-system simulations with network and device modeling. This makes gem5 a valuable tool for evaluating future accelerators, processor cores, system-on-chips, hardware/OS/network interactions, and heterogeneous systems [20].

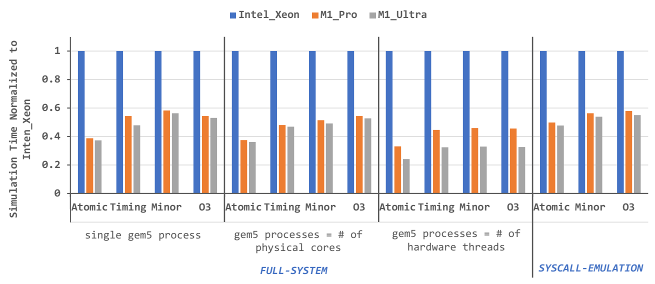

Fig. 1: Geometric mean of the normalized simulation time when running PARSEC and SPLASH-2x workloads on gem5. gem5 runs on `Intel_Xeon`, `M1_Pro`, and `M1_Ultra`. Atomic, Timing, Minor, and O3 are gem5 CPU types.

One observation that motivated this work is the drastic difference in simulation speed when running gem5 on different server platforms. Fig.1 shows the geometric mean of the simulation time of executing gem5 on a MacBook Pro (`M1_Pro`) and a Mac Studio (`M1_Ultra`), normalized to the simulation time on a Dell server equipped with Xeon Gold Scalable CPUs (`Intel_Xeon`) across all nine (9) PARSEC and SPLASH-2x workloads. Both MacBook and Mac Studio are equipped with Apple M1 chips. More information on the workloads, simulated system configuration, physical hardware configurations, and experimental methodology are provided in Sec.III. We run gem5 in full-system (FS) and system-call emulation (SE) modes. An important parameter in the tests performed is the number of processes simultaneously running on each platform. In the left most and right most sub-graphs of Fig.1, we run a single gem5 process on the host server, while in the middle sub-graphs, we co-run one gem5 process per physical core and one gem5 process per hardware thread. There are 4, and 16 performance cores in `M1_Pro` and `M1_Ultra`, and 20 physical cores and 40 hardware threads on `Intel_Xeon`. Therefore for co-running scenarios, we launch 4 (`M1_Pro`), 16 (`M1_Ultra`), 20 (`Intel_Xeon` for "gem5 processes = # of physical cores" with SMT off configuration), and 40 (`Intel_Xeon` for "gem5 processes = # of hardware threads" with SMT on configuration) gem5 processes.

As shown in the figure, regardless of whether SMT is turned on or off for `Intel_Xeon` (*it is worth noting that `M1_Pro` and `M1_Ultra` does not support hardware multithreading*), simulation mode (full system vs. system-call emulation) or simulation detail (Atomic vs. Timing or In-order vs. Out-of-Order), M1 platforms consistently deliver lower simulation time. This applies to different benchmarks simulated on gem5 as illustrated in Fig.1. The simulation speed of M1 platforms is even higher when co-running multiple gem5 processes. As depicted, running gem5 on an `M1_Ultra` is up to 4.15× faster compared with execution on a high-end Xeon server. We see on average ∼47% performance improvement on `Intel_Xeon` with SMT disabled. That is, the simulation time of running 20 gem5 processes (with SMT disabled) is ∼47% less than running 40 gem5 processes (SMT enabled). As we will discuss in Sec. IV this is expected since gem5 is sensitive to L1 cache size and disabling SMT will reduce cache contention, thus improving the overall simulation speed.

Motivated by the huge speedup gains by just running gem5 on a different platform, we set out to profile the execution of gem5 on both Intel and M1 platforms to shed light on gem5's execution bottlenecks. Many of our insights from profiling gem5 can be applied to other architectural simulators or even simulators in different fields.

III. Methodology

In terms of simulation configurations, we change the CPU type, number of CPUs, and memory size. We use the following CPU types:

AtomicSimpleCPU (Atomic): CPU type with CPI = 1 where memory accesses are atomic and completed without modeling any contention or queuing delays.

TimingSimpleCPU (Timing): CPU type with CPI = 1 where memory accesses are modeled in detail considering the queuing delays and resource contentions in the memory and interconnect.

In-order CPU (Minor): In-order or Minor CPU models a fixed pipeline with strict in-order instruction execution. Minor CPU uses the detailed timing memory model for accessing memory.

Out-of-order CPU (O3): O3 CPU models an out-of-order superscalar loosely based on the Alpha 21264 core. O3 CPU uses the detailed timing memory model for accessing memory.

Simple CPUs are used for fast-forwarding simulation, warming up caches, or for studies that do not require detailed CPU modeling. In-order and out-of-order CPU models are used for detailed microarchitectural studies. Table I shows the processor configuration for each CPU type. We used Linaro 7.5.0 toolchain for SPLASH-2x and an Ubuntu 18.04 disk image for PARSEC workloads, respectively. We use Linux kernel 5.4.0 and ARM ISA for full-system simulations [27].

TABLE I: Base Hardware Configuration on FireSim

Parameters	Value
Core Frequency	4GHz
Number of Cores	4 Cores
Superscalar	8-width wide
ROB/IQ/LQ/SQ Entries	192/64/32/32
Int & FP Registers	128 & 192
Branch Predictor/BTB Entries	TournamentBP/4096
Cache: L1I/L1D	48KB(I), 32KB(D)
DRAM	2GB, DDR3-1600-8x8
Operating System	Linux Linaro (kernel 5.4.0)

We simulate the following workloads on gem5:

- ***Boot-Exit***: Boot Linux in FS mode and immediately exit. Note that *`M1_Pro` and `M1_Ultra` cannot take readable checkpoints so we use them to recover from checkpoints taken by `Intel_Xeon`.*
- ***PARSEC***: We execute `Canneal`, `Blackscholes`, `Dedup`, and `streamcluster` within the mainline PARSEC 3.0 benchmark and `water_nsquared`[2], `water_spatial`, `ocean_ncp`, `ocean_cp`, and `fmm` apps within SPLASH-2x [28] benchmark suite. The benchmark input size used for all workloads is `simmedium`.
- ***C++ program***: Because FireSim is orders of magnitude slower than real hardware (a gem5 simulation that completely executes in 2.34 seconds on a quad-core `Intel_Xeon` host results in a slowdown of $\sim$118$\times$ on Firesim), we run a simple algorithm called `Sieve_of_Eratosthenes` on the simulated node on FireSim to evaluate the performance of gem5.

[2]In this paper, Top-Down microarchitectural analysis was carried out using water_nsquared as a representative workload from PARSEC benchmark suite

TABLE II: Evaluation platforms

Platform	Dell Precision 7920	Apple Macbook	Apple MacStudio
Config Name	`Intel_Xeon`	`M1_Pro`	`M1_Ultra`
SoC	Xeon Gold 6242R	M1	M1 Ultra
micro-architecture	Cascade Lake	Firestorm(P) + Icestorm(E)	
Cores	20C/40T	P:4C/4T + E:4C/4T	P:16C/16T + E:4C/4T
Max Freq	3.1GHz (4.1GHz TB)	3.2GHz(P), 2GHz(E)	3.2GHz(P), 2GHz(E)
L1 (per-core)	32KB(I) + 32KB(D)	P:192KB(I) + 128KB(D) E:128KB(I) + 64KB(D)	
L2	20MB	P:12MB + E:4MB	P:48MB + E:8MB
L3	35.75MB	8MB	96MB
Cacheline	64B	128B	128B
Memory	96GB, DDR4-2933	8GB, LPDDR4X-4266	64GB, LPDDR5-6400
DRAM BW	141 GB/s	68 GB/s	819.2 GB/s
Single core BW (unloaded)	45 GB/s	58 GB/s	58 GB/s
DRAM Latency	96ns	97ns	97ns
VM page size	4KB	16KB	16KB

We used the Intel VTune profiler [29] to access the processor performance counters and perform the Top-Down microarchitectural analysis [30]. To collect performance counters on the `Intel_Xeon` CPUs, perf was used [31] . We also profile Apple M1 CPUs by reading performance counters from the privileged level [32]. We modify the main simulation loop function in gem5 to read the performance counters for each execution.

Experiments are run on three platforms. Table II summarises the configuration of these three platforms [33]. We refer to these platforms using their configuration name (Config Name in Table II) throughout the paper: `Intel_Xeon`, `M1_Pro`, `M1_Ultra`.

Since gem5 is a single-threaded application, its microarchitectural behavior can be directly compared to single-threaded CPU benchmarks. For this comparison, we choose a mix of 3 benchmarks from the SPEC 2017 benchmark suite [34]. The SPEC workloads that we choose are:

- `525.x264_r` has been observed to have the highest IPC of all benchmark in SPEC 2017 suite [35].
- `531.deepsjeng_r` has a large memory footprint and has been observed to have the highest L3 cache miss rate among other SPEC benchmarks [36].
- `505.mcf_r` is chosen due to its high front-end and back-end stalls resulting from cache misses, and branch misprediction. `505.mcf_r` has the lowest IPC of all benchmarks in SPEC 2017 suite [35].

Note that *we run SPEC benchmarks on bare metal hardware, not on gem5*. We only use SPEC benchmarks as a reference to compare with gem5's top-down profile in Sec.IV-A.

Using FireMarshal [37], we run gem5 as a workload on Firesim [17] for profiling purposes. In our study, we execute gem5 in system-call emulation mode on a chipyard SoC design [38]. Our base hardware configuration on Firesim is a quad-core Rocket chip, which is a RISC-V open-source CPU with a single-

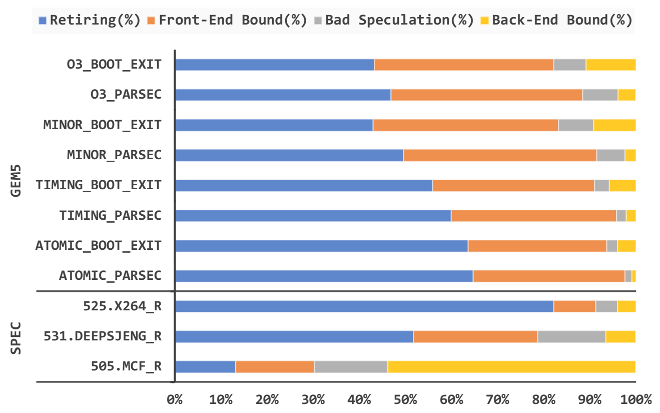

Fig. 2: Top-level bottleneck breakdown of gem5 (top) and SPEC (bottom) running on `Intel_Xeon` platform. Note that we include the SPEC workload analysis to provide a reference for comparison. The SPEC benchmarks are running on native hardware.

issue, in-order pipeline processor. Our design incorporated an 8KB 2-way set associative iCache, an 8KB 2-way set associative dCache, and a 512KB 8-way SiFive set associative cache, each with a 64-byte block size. We also use a host FPGA frequency of 140MHz and a DDR3FRFCFSLLC4MB config fragment to emulate the memory model. We build different custom hardware designs by varying the cache configs which we discuss in Sec. V-B.

IV. Profiling gem5

In this section, we profile gem5 simulating various configurations and report microarchitectural statistics to demystify the execution inefficiencies of gem5 as an application. Our goal is to gain some insights into the execution profile of gem5 to set the stage for future targeted hardware-level, system-level, and application-level optimizations that will make gem5 run faster.

A. Top-Down Analysis on Intel Xeon Platform

We use VTune to perform a Top-Down microarchitectural performance analysis [30] of gem5 when simulating different workloads with varying configurations. Top-Down analysis split the machine cycles into four categories: *retiring*, *front-end bound*, *bad speculation*, and *back-end bound*. Ideally, we want every cycle to be categorized as *retiring*, which is the only category in which the CPU performs useful work. A cycle is considered to be *front-end bound* if the fetch and decode units (i.e., front-end of the out-of-order processor) cannot supply sufficient μ-ops for the back-end. The main culprits for front-end bound cycles are iCache/iTLB misses and inefficiencies in the instruction decoders. A cycle is considered to be back-end bound when the processor is stalled because there are not enough resources in the back-end. This would occur when the load/store queue is full or the functional units are all busy. Lastly, a cycle is considered to be *bad speculation* when a cycle is lost due to running miss-speculated instructions or recovering from previous bad speculation.

Fig.2(top) shows the top-level profiling results categorizing the CPU cycles spent executing gem5 with different CPU

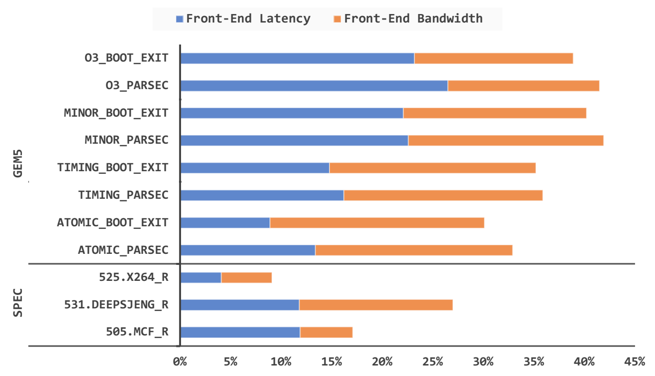

Fig. 3: Front-end latency bound cycles breakdown for gem5 (top) and SPEC (bottom) running on `Intel_Xeon`.

types. To have a reference for comparison, we also run three SPEC 2017 benchmarks with diverse characteristics on the `Intel_Xeon` platform and show their Top-Down analysis at the bottom of Fig.2. Refer to Sec.III for more information on the choice of SPEC benchmarks. As shown, 43.5∼64.7% of cycles retire instructions across different gem5 simulations. This is a relatively high retiring percentage compared to conventional workloads. As shown in the figure, the retiring cycle percentage for SPEC 2017 benchmarks are between 13.2∼82.2%. However, gem5's *front-end bound* cycles are much higher while the *back-end bound* cycles are lower compared with SPEC. `505.mcf_r`, which is a memory-intensive workload. It has 53.7% of back-end bound cycles while gem5 workloads only spent 0.9%∼11.3% of their cycles stalling for back-end. This is expected since gem5's dynamic working set increases very slowly while simulating different workloads as gem5 is orders of magnitude slower than real hardware. Moreover, the simulated memory size is often small and limited to a few gigabytes, which is not even fully touched by the simulated workload. The small dynamic working set size and temporally slow memory access to this working set results in predictable data cache accesses from gem5 that can be efficiently captured by the hardware prefetchers or overlapped in the out-of-order engine of the modern processors.

Hyper-scale workloads such as web-search, web-serving, and video processing are known to have large instruction cache footprint and thus are considered to be front-end bound as their *front-end bound* cycles are 2∼3× more than those in typical SPEC benchmarks [39] (in the 15∼30% range across various workloads). Looking at Fig.2(top), the front-end bound cycles for gem5 are in the 30.1%∼41.5% range, which is even higher than that of hyper-scale workloads. Next, we present a breakdown of performance events impacting front-end bound stall cycles to find out why such a large number of cycles are spent waiting for the front-end to supply instructions.

Figure 3 shows the classification of the front-end bound cycles between front-end bandwidth and latency. The main reasons for bandwidth- and latency-bound cycles are inefficiencies in instruction decoding and iCache/iTLB misses, respectively. As shown in Fig.3(top), simpler CPU models are more skewed toward bandwidth-bound and as the level of CPU detail increases, the front-end becomes more latency-bound.

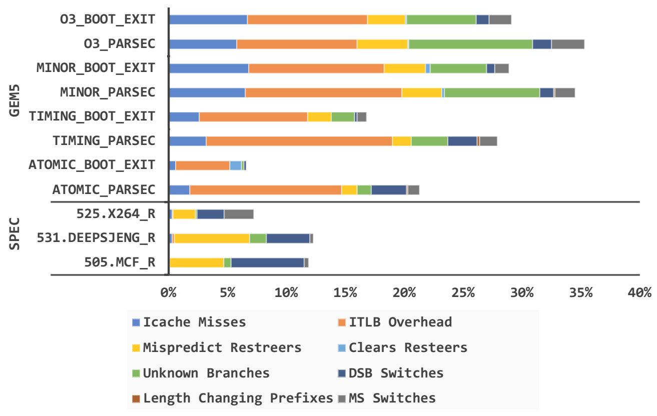

Fig. 4: Front-end latency bound cycles breakdown for gem5 (top) and SPEC (bottom) running on `Intel_Xeon`.

This can be explained by the fact that as the complexity of the CPU model increases, gem5 touches more simulation object binaries for processing each event. Therefore, the instruction cache footprint increases with the CPU model complexity, and consequently, gem5 becomes more front-end latency-bound. Compared with SPEC, gem5 is more front-end bandwidth-bound. Next, we zoom into the front-end latency and bandwidth breakdown to better understand the bottlenecks in the front-end when executing gem5 on `Intel_Xeon`.

Figure 4(top) plots the breakdown of front-end latency-bound cycles. As illustrated in the figure, the O3 and Minor CPUs have up to 11× higher iCache misses compared with Atomic CPU simulations. Interestingly, stalled cycles due to iTLB misses are high across all gem5 executions. On the other hand, SPEC benchmarks are neither iCache nor iTLB bound. Along with iCache and iTLB overheads, we see a huge increase in the branching-related overhead when using O3 and Minor CPUs. The aggregated branching overhead for O3_PARSEC and Minor_PARSEC (sum of Mispredict Resteers, Clear Resteers, and Unknown Branches) is 6.0× and 4.7× higher than that of ATOMIC_PARSEC. As shown in the figure, by using more detailed CPUs, the percentage of unknown branches significantly increases. The high branch overhead of detailed gem5 simulation occurs because increasing the CPU model's complexity initiates more function calls, parameter checks, and event generation and activation. These, in turn, increase the branch density of the code, contribute to the large branch overhead, and increase the number of hard-to-predict branches. For SPEC benchmarks, the branching category contributes to the majority of the front-end latency-bound cycles. Mispredict Resteers and Unknown Branches alone contribute to 43.5%∼73.6% of total front-end latency-bound cycles in SPEC benchmarks.

Figure 5(top) shows the breakdown of bandwidth-bound cycles. Interestingly, between 92∼97% of the front-end bandwidth-bound cycles are limited due to waiting for MITE (Micro-Instruction Translation Engine), and only less than 7% are bounded by the DSB (know as Decoded iCache or μOp Cache) μOp supply. μOp cache is a small memory structure in the decoder unit that holds hot μOp traces. μOp cache works for codes with lots of instruction reuse and loops, which

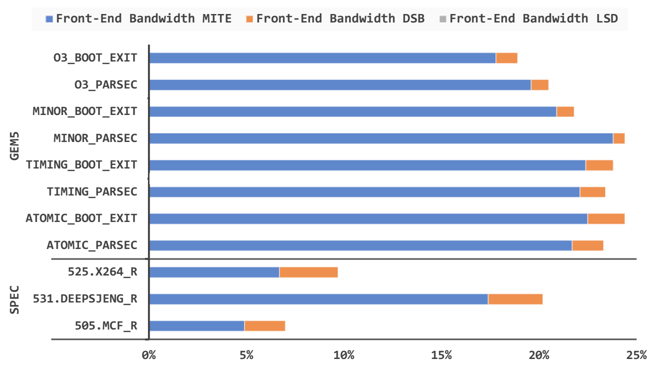

Fig. 5: Front-end bandwidth bound cycles breakdown for gem5 (top) and SPEC (bottom) running on `Intel_Xeon`.

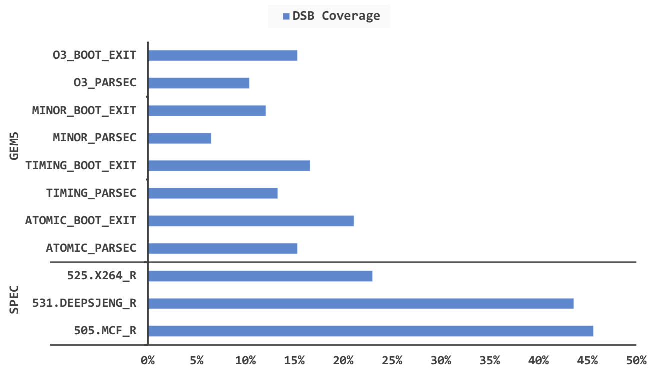

Fig. 6: DSB (μOp Cache) coverage of gem5 (top) and SPEC (bottom) running on `Intel_Xeon`.

are both rare in gem5. The irregularity in the gem5's code results in a lot of pressure on the instruction decoder to supply enough instructions to the back-end. Compared to gem5, as shown in Fig.5(bottom), when running SPEC, more of the front-end bandwidth-bound cycles are categorized under DSB. This is because the DSB coverage for regular applications is often very high. Fig.6 compares the DSB coverage of gem5 and SPEC benchmarks running on `Intel_Xeon`. As shown, the DSB coverage of gem5 is much lower than that of SPEC, regardless of the CPU type or workload. This puts pressure on the decoder and thus, the MITE stall cycles are high for gem5 simulations.

B. *Profiling gem5 on M1*

In the previous subsection, we performed a detailed top-down analysis of gem5 running on a `Intel_Xeon` platform and compared its behavior against conventional SPEC benchmarks. We understand that the front-end of the `Intel_Xeon` is the bottleneck in running gem5 simulations. There are many iCache and iTLB misses, and the instruction decoder of complex x86 instructions cannot feed enough μops to the out-of-order back-end. We perform the analysis while running gem5 simulations using three CPU types (Atomic, Timing, and O3), then we execute `water_nsquared` on gem5.

Figure 7 compares the average instruction per cycles (IPC) and CPU stalled cycles of `Intel_Xeon`, `M1_Pro`, and `M1_Ultra` when running gem5 simulations. IPC of `M1_Pro`

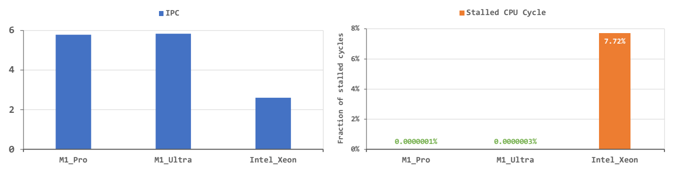

Fig. 7: IPC (left) and percentage of stalled CPU Cycles (right) for `Intel_Xeon`, `M1_Pro`, `M1_Ultra` running gem5.

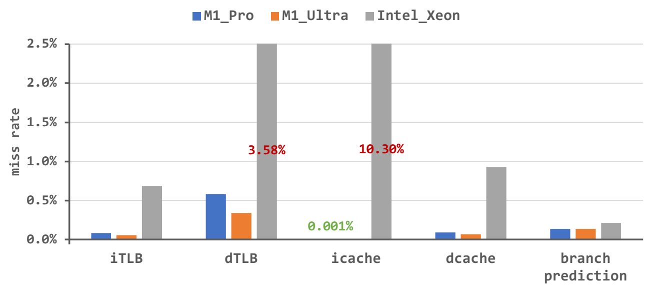

Fig. 8: Performance comparison of TLB, L1 cache, and branch predictor of `Intel_Xeon`, `M1_Pro`, and `M1_Ultra`.

and `M1_Ultra` are 2.22× and 2.24× higher than that of `Intel_Xeon`, respectively. This margin in IPC values across both platforms reflects on the simulation time differences illustrated in Fig.1. Unsurprisingly, the time percentage that `Intel_Xeon` is stalled is much higher than that of `M1_Pro` and `M1_Ultra`.

Figure 8 compares the TLB, L1 cache, and branch prediction performance of `Intel_Xeon`, `M1_Pro`, and `M1_Ultra` when running gem5 simulations. As shown in the figure, the iTLB, dTLB, and L1 cache miss rates of M1 platforms are much lower than that of `Intel_Xeon`; iTLB and dTLB miss rates of `Intel_Xeon` are 11.7× and 10.5× higher than that of `M1_Ultra`, respectively. We believe the main cause of the performance difference between M1 platforms and `Intel_Xeon` reported in Fig.1 are the TLB and L1 caches. Looking at Table I, the performance cores in both `M1_Pro` and `M1_Ultra` have 192KB iCache and 128KB dCache, while `Intel_Xeon` has 32KB iCache and 32KB dCache. This is 6× and 4× larger iCache and dCache for M1 platforms, respectively.

Although there is no information on the associativity of the L1 cache of M1 platforms, we can reverse engineer the number of ways assuming that the L1 is implemented as a virtually-indexed, physically tagged (VIPT) cache. In VIPT caches, the total capacity of a single way cannot exceed the virtual memory page size in order to overlap the TLB access (address translation) with indexing into the cache [40]. Since M1 uses 16KB virtual memory page sizes, the iCache and dCache associativity should be 12 and 8, respectively. The number of ways for the 32KB icache and dcache of `Intel_Xeon` is 8 ways. Therefore, the 10.1× ∼13.4× reduction in dCache miss rate for M1 platforms shown in Fig.8 is mostly due to the reduction in capacity and compulsory misses (4× higher capacity, 2× larger cache line size).

We also notice that the branch prediction accuracy of M1 platforms is higher than that of `Intel_Xeon`. As shown in

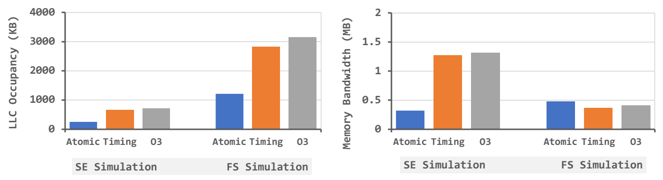

Fig. 9: LLC occupancy and memory bandwidth utilization of gem5 running with different configurations and operating modes on `Intel_Xeon`.

Fig.8, the branch misprediction rate of `Intel_Xeon` is 0.22% while both M1 platforms have ∼0.14% branch misprediction rates. In Sec.V-B we run gem5 on FireSim and study the impact of changing the L1 and L2 cache configurations of the host (simulation server) on gem5's performance. What is clear is that the combination of using larger cache lines (64B vs. 128B), larger virtual memory page size (4KB vs. 16KB), and larger L1 caches [3] (32KB vs. 128KB) dramatically improves L1 and TLB performance in M1 platforms. The performant TLB, L1, and branch predictor results in higher IPC, and in turn, higher simulation speed for `M1_Pro` and `M1_Ultra` when compared to `Intel_Xeon` (Fig.7 and Fig.1).

Figure 9(left) shows the LLC occupancy per gem5 process and Fig.9(right) shows the DRAM bandwidth utilization of gem5 when running simulations with different CPU models in Full-System (FS) and System-call Emulation (SE) modes on `Intel_Xeon`. Unfortunately, we were not able to find L2, LLC, or DRAM-related performance counters on M1 platforms. Therefore, we could not include M1-related information in Fig.9. As shown in Fig.9 (right), surprisingly, the DRAM bandwidth utilization of gem5 is negligible regardless of whether it is running in FS or SE mode. Such low DRAM bandwidth utilization suggests that gem5's data set size fits in the last-level cache (LLC). Fig.9 (left) plots the LLC occupancy of a single gem5 process running on `Intel_Xeon`. As shown, the LLC occupancy increases with the detail level of simulation, and a gem5 simulation with O3 CPU has the largest instruction and data footprint compared to simulation with Atomic and Timing CPUs. The LLC occupancy of a single gem5 process is between 255KB∼3.1MB.

V. Sensitivity Analysis of Simulation Speed

In this section, we leverage our insights from Sec. IV to perform a sensitivity analysis for the simulation speed of gem5. We divide the analysis into *systems* and *architecture* sensitivity. Under the *systems* analysis, we study the sensitivity of simulation speed to several systems and compiler parameters that do not require changes to the server hardware or gem5 application. Under architecture analysis, we study the sensitivity of simulation speed to the size and associativity of the L1 and L2 caches of the simulation server. Since such analysis requires changes to the hardware, we run gem5 on FireSim and configure FireSim to simulate a host server with various cache configurations. We run unmodified gem5 on FireSim.

[3]The larger virtual memory page size enables the implementation of low-associativity, large VIPT L1 caches in M1 platforms as explained earlier.

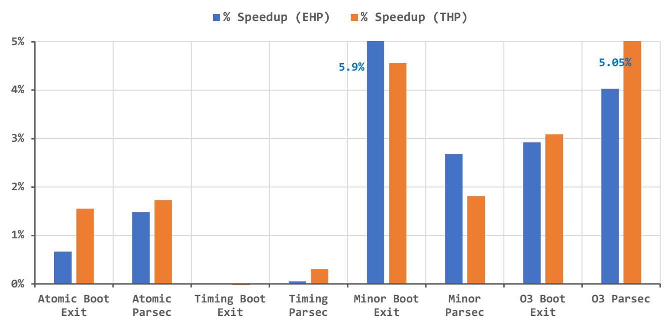

Fig. 10: Performance gain from enabling huge pages for gem5 simulations running on `Intel_Xeon`.

A. Sensitivity to System Configurations

As discussed in detail in Sec.IV-A, due to the large instruction footprint of gem5, we observe many stalled cycles due to iTLB misses while running gem5 simulations. A simple solution to the iTLB misses is to use huge pages to back gem5 code/text. We explored two ways to back gem5 code/text regions with huge pages: Transparent Huge Pages (THP) and Explicit Huge Pages (EHP). Linux supports Transparent Huge Pages (THP) [41] which is a kernel feature that provides dynamic huge page allocations at application runtime. The current Linux THP implementation only works with anonymous memory mappings (i.e., the memory that is not backed by the file system such as implicit memory allocations on the heap and stack) and tmpfs/shmem. We utilize an open-source library by Intel [42] to back gem5's code segment with transparent/explicit huge pages. By invoking a few API calls at the beginning of gem5 runtime, the library automatically remaps a subset of gem5's code to 2MB huge pages [43]. We also explored the use of libhugetlbfs library package [44], which requires that gem5 be recompiled so the binary is aligned at huge page boundaries. libhugetlbfs automatically backs the code, data, heap, and shared memory segments with explicitly allocated huge pages when invoked with requisite parameters. However, our experiments show an abysmal improvement in simulation time compared with Intel iodlr. We suggest this is a result of a sub-optimal gem5 binary layout.

As shown in Fig.10, using huge pages to back the code of gem5 improves simulation speed by up to 5.9%. The benefits from using huge pages are low for simple CPU models (i.e., Atomic and Timing CPUs), while the benefits for more detailed CPU models are higher. This is expected because the code footprint of simple CPUs is smaller than that of detailed CPU models. This is in line with our discussion in Sec.IV-A and Fig.3 which illustrates that the simulation of simpler CPUs is less front-end latency bound compared to detailed CPUs.

We do not see any specific pattern in the performance of EHP and THP. For some configurations, EHP performs better than THP. Fig.11 shows the improvement in iTLB overhead and retiring cycles when backing gem5's code with THP. As shown in Fig.11(top), using THP significantly reduces the iTLB overhead for Minor and O3 simulations. On average, THP reduces the iTLB overhead by 63%. The improvement in iTLB overhead results in 3∼7% improvement in the number of retiring clock cycles in the CPU pipeline for Minor and O3 CPU simulations (Fig.11(bottom)).

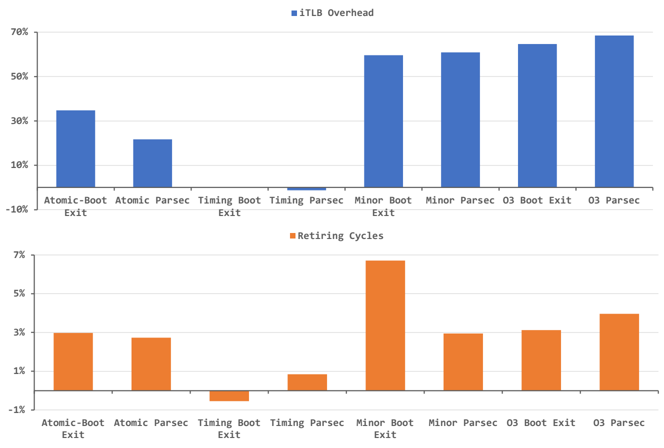

Fig. 11: Improvement in iTLB overhead and retiring cycles when backing gem5 code with transparent huge pages.

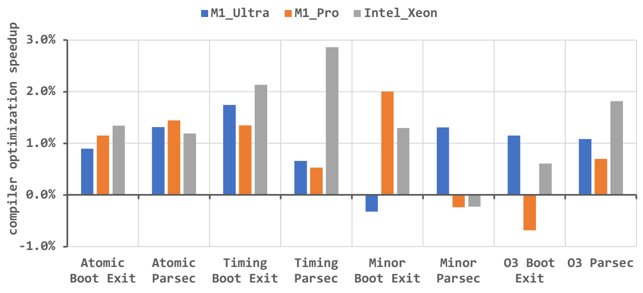

Fig. 12: Improvement in gem5 simulation speed when applying compiler optimizations on `Intel_Xeon`, `M1_Pro`, and `M1_Ultra` platforms.

Next, we study the impact of compiling gem5 using the "-O3" flag passed to the GNU G++ compiler. We modified the `scons` script to compile gem5 with a higher level of compiler optimization (i.e, "-O3" flag). Though we used `gem5.opt`, we still notice a reduction in the size of the resulting binary and in the simulation time. Fig.12 compares the simulation speed up when using a gem5 binary that is compiled with "-O3" flag compared with baseline gem5 compiled without the optimization flag. On average, this simple change in the build process results in 1.38%, 0.98%, and 0.78% speedup for `Intel_Xeon`, `M1_Pro`, and `M1_Ultra` platforms. "-O3" flag only performs static compile time optimizations and thus there is a possibility for hurting the application speed after applying the optimizations. We see a few instances of such cases in Fig.12.

Lastly, we study the impact of CPU frequency on gem5 speed. Fig.13 shows how simulation time changes when running gem5 on `Intel_Xeon` operating at various frequencies. The simulation times in Fig.13 are normalized to the run with 3.1GHz frequency. As expected, reducing CPU frequency from 3.1GHz to 1.2GHz increases the simulation time by 2.67×. This shows a linear increase in simulation time with a reduction in CPU frequency.

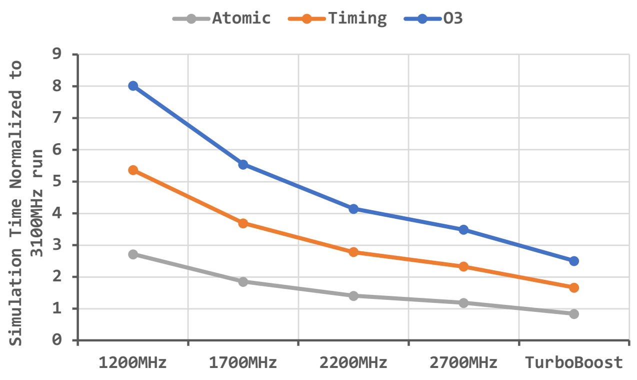

Fig. 13: Normalized gem5 simulation time when changing CPU frequency and enabling Turbo Boost on `Intel_Xeon` platform. The simulation times are normalized to the baseline CPU running at 3100MHz without TurboBoost.

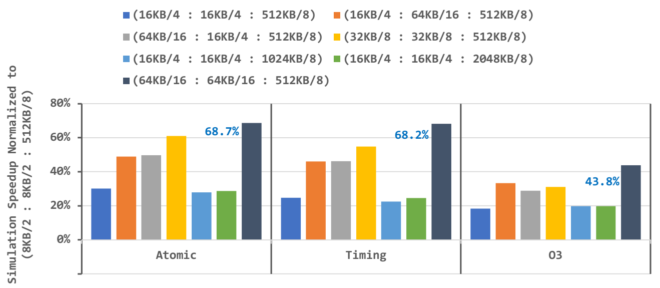

Fig. 14: Simulation speedup when running gem5 on FireSim with varying cache configurations.

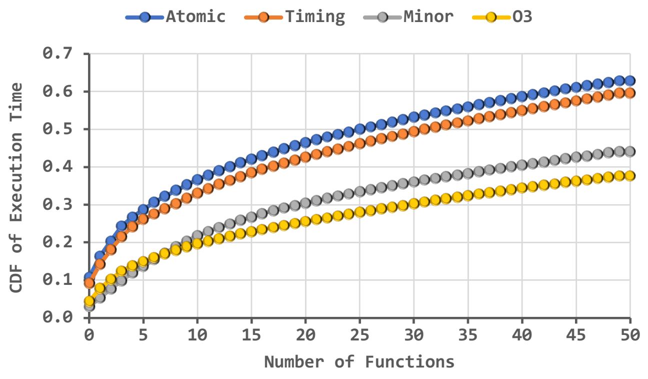

Fig. 15: Top 50 hottest functions in gem5 when simulating a PARSEC workload with different CPU types.

B. Sensitivity to Architectural Configurations

In this subsection, we run gem5 on FireSim and change the L1 and L2 configurations of the O3 core in FireSim to study gem5's sensitivity to the cache configuration of the simulation server. As mentioned earlier, since FireSim is much slower than real hardware, we run a simple C++ application on gem5 and do not run PARSEC.

Figure 14 compares the simulation time of gem5 running on a server with various L1 and L2 configurations. The cache line size and virtual memory page size in the simulated node in FireSim are 64B and 4KB, respectively. Since the L1 cache is a VIPT cache, to increase the L1 size we only increase the associativity and keep the number of sets fixed at 64 to overlap TLB access and L1 cache indexing. We use the following format to represent different L1 and L2 configurations in Fig.14: (iCache size/iCache associativity : dCache size/dCache associativity : L2 size/L2 associativity).

As illustrated in Fig.14, increasing the size of both iCache and dCache are critical in improving the simulation speed. The simulation times are normalized to a baseline configuration with 8KB 2-way set associative iCache/dCache and a 512KB 8-way set associative L2 cache. Increasing iCache and dCache size from 8KB to 16KB reduces Atomic, Timing, and O3 simulation time by 30%, 25%, and 18%. On the other hand, doubling L2 cache size from 1MB to 2MB has almost no impact on the simulation time. The best-performing configuration is the last configuration where we keep L2 the same size as the baseline and configure both iCache and dCache as 64KB 16-way set-associative caches (64KB/16 : 64KB/16 : 512KB/8). This configuration improves simulation speed by 68.7%, 68.2%, and 43.8% for Atomic, Timing, and O3 simulations, respectively. We notice that gem5 simulations with O3 CPU benefit less than simpler CPU models from increasing L1 cache size. We suspect that the TLB bottleneck in detailed simulations limits the benefits of the larger L1 size.

VI. Discussion of Future Work

Our detailed microarchitectural analysis revealed the bottlenecks in gem5 execution. The fact that changing the physical platform can result in significant simulation speedup motivates us to think about developing specialized computing platforms for running architectural simulations. One potential area to explore is to offload simulation entirely or partially to a hardware accelerator.

For an application to be qualified for hardware acceleration, the application needs to be (1) widely used and (2) stable with no structural changes over time. gem5 satisfies the first requirement as it is a popular application with a growing user base in both academia and industry. Although gem5 is constantly changing, its core, which is the event queue and event scheduler has been the same for many years, and will probably remain the same in the future. Therefore, gem5 satisfies the second requirement as well.

Figure 15 shows the cumulative distribution function (CDF) of the CPU time of the 50 hottest functions executed in gem5 simulating different CPU types. As shown, *there is no killer function inside the gem5 source code whose optimization would significantly improve the simulation time.* The hottest function in Atomic, Timing, Minor, and O3 CPU types contribute to 10.1%, 8.5%, 2.9%, and 4.2% of the total simulation time, respectively. As we increase the complexity of the CPU, the CDF of individual function execution time gets flatter; meaning that the hotness of individual functions gets lower. This is not surprising since increased simulation complexity causes more simulation objects to get activated in each event to more accurately model the complexity of the hardware. Therefore, more diverse functions get called when simulating with O3 CPU type compared with simpler CPU models. The total number of functions called throughout the simulation for the results shown in Fig.15 are 1602, 2557, 3957, and 5209 for Atomic, Timing, Minor, and O3 CPU types, respectively.

Since there is no killer function, accelerating even several gem5 functions in hardware would not provide a significant performance improvement. Therefore, the results of Fig.15 suggest that building an off-chip hardware accelerator for gem5 is probably not an option. Instead, hardware acceleration should be at a finer granularity and tightly coupled with the CPU. The comparison of `Intel_Xeon` and M1 platforms revealed that even a general-purpose CPU with some fine-tuning of architectural and system parameters can significantly improve gem5 performance. Detailed basic-block analysis of gem5 source code is required to identify commonly used operations and data structures to map them to specialized, complex instructions. The open-source RISC-V ISA facilitates the development of such a specialized CPU. Designing such a specialized CPU for accelerating event-driven simulation is an interesting future research direction. However, a short-term solution is to utilize the configurability of current servers at the system- and compiler-level optimizations to improve the simulation execution.

VII. Related Work

Acceleration of Simulation FireSim [17] is an FPGA Architecture Model Execution (FAME) simulator [45]. FireSim uses FPGAs to implement the complete RTL of a target system to model the timing of future hardware. The timing model can be decoupled from the real design such that multiple host FPGA clock cycles are used to model one target clock cycle when modeling complex logic. Although FAME simulators significantly speed up the simulation speed, developing new models on them is time-consuming, error-prone, and inflexible. Therefore FAME cannot replace software-base simulation. Parallel Discrete Event Simulation (PDES) technique is used to simulate different components in parallel and conservatively or optimistically synchronize them in fixed intervals called quantum [46], [9], [47]. Using conservative PDES for parallelizing the simulation of on-chip resources has diminishing returns as the overhead of frequent synchronization offsets the benefits of parallel execution. Therefore, PDES simulators such as SST [47] only operate at larger component levels since the speed of modeling individual components will become a bottleneck in the overall simulation.

Sampling techniques and using hardware virtualization support for fast-forwarding simulations are widely used for improving the speed and accuracy of architectural simulation [10], [11], [12], [13], [14]. Such techniques are orthogonal to improving the speed of detailed simulation.

Top-Down Microarchitectural Analysis. There is a large body of work on profiling applications and performing Top-Down microarchitectural analyses on various applications such as data analytics and cloud applications [48], [49], hyper-scale services [39], SPEC benchmark [30], [35], [50], web search [51], network stack [52], data-intensive applications [53], [54], [55], video transcoding [56], graph applications [57], network fuctions [58], and many more application domains. However, no previous work has performed a detailed Top-Down microarchitectural analysis to profile the execution of a software-based architectural simulator.

VIII. Conclusion

There has been no work characterizing the execution bottlenecks in gem5 even though it is considered one of the most versatile and slowest architectural simulators and has a huge active user community. In this work, we profiled the performance characteristics of gem5 and demystified the inefficiencies of gem5 simulations. Our detailed Top-Down microarchitectural analysis reveals three main bottlenecks in gem5 execution: (1) high iCache and iTLB misses, (2) high branch resteer overheads, and (3) extremely low μOp cache utilization when running on an Intel Xeon CPU. These bottlenecks are the result of huge code size, cold code execution, extensive use of virtual functions, and polymorphism throughout the gem5 source code. We observe that running gem5 on an Apple M1 MacBook reduces simulation time by up to $3\times$times compared to a high-end Xeon server. Our profiling results reveal that the larger L1 cache size along with the use of a larger virtual memory page size leads to such performance improvement for gem5. This work is the first step towards better understanding the characteristics of detailed, software-based architectural simulation and developing optimized server solutions for accelerating the simulation of future computer systems.

Acknowledgement

This work was supported in part by grants from National Science Foundation (CNS-2213807) and ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

References

[1] "John hennessy and david patterson 2017 acm a.m. turing award lecture," https://www.youtube.com/watch?v=3LVeEjsn8Ts, 2017.

[2] A. Akram and L. Sawalha, "A survey of computer architecture simulation techniques and tools," *IEEE Access*, vol. 7, 2019.

[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, 2011.

[4] W. Heirman, T. Carlson, and L. Eeckhout, "Sniper: Scalable and accurate parallel multi-core simulation," in *8th International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems (ACACES-2012)*. High-Performance and Embedded Architecture and Compilation Network of ..., 2012.

[5] A. Patel, F. Afram, and K. Ghose, "Marss-x86: A qemu-based micro-architectural and systems simulator for x86 multicore processors," in *1st International Qemu Users' Forum*, 2011.

[6] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, 2013.

[7] K. Asanović and D. A. Patterson, "Instruction sets should be free: The case for risc-v," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2014-146*, 2014.

[8] SST, "Sst simulation," ://sst-simulator.org/.

[9] A. Mohammad, U. Darbaz, G. Dozsa, S. Diestelhorst, D. Kim, and N. S. Kim, "dist-gem5: Distributed simulation of computer clusters," in *Performance Analysis of Systems and Software (ISPASS), 2017 IEEE International Symposium on*. IEEE, 2017.

[10] A. Sandberg, N. Nikoleris, T. E. Carlson, E. Hagersten, S. Kaxiras, and D. Black-Schaffer, "Full speed ahead: Detailed architectural simulation at near-native speed," in *2015 IEEE International Symposium on Workload Characterization*, 2015.

[11] G. Borgström, A. Sembrant, and D. Black-Schaffer, "Adaptive cache warming for faster simulations," in *Proceedings of the 9th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3023973.3023974

[12] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proceedings of the 30th annual international symposium on Computer architecture*, 2003.

[13] E. Perelman, G. Hamerly, M. Van Biesbrouck, T. Sherwood, and B. Calder, "Using simpoint for accurate and efficient simulation," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, 2003.

[14] N. Nikoleris, L. Eeckhout, E. Hagersten, and T. E. Carlson, "Directed statistical warming through time traveling," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3352460.3358264

[15] J. Chen, M. Annavaram, and M. Dubois, "Slacksim: A platform for parallel simulations of cmps on cmps," *SIGARCH Comput. Archit. News*, vol. 37, jul 2009. [Online]. Available: https://doi.org/10.1145/1577129.1577134

[16] M. Alian, D. Kim, and N. S. Kim, "pd-gem5: Simulation infrastructure for parallel/distributed computer systems," *IEEE Computer Architecture Letters*, vol. 15, 2016.

[17] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanovic, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[18] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "Hasim: Fpga-based high-detail multicore simulation using time-division multiplexing," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, 2011.

[19] J. Wawrzynek, D. Patterson, M. Oskin, S.-L. Lu, C. Kozyrakis, J. C. Hoe, D. Chiou, and K. Asanovic, "Ramp: Research accelerator for multiple processors," *IEEE Micro*, vol. 27, 2007.

[20] J. Lowe-Power, A. M. Ahmad, A. Akram, M. Alian, R. Amslinger, M. Andreozzi, A. Armejach, N. Asmussen, B. Beckmann, S. Bharadwaj *et al.*, "The gem5 simulator: Version 20.0+," *arXiv preprint arXiv:2007.03152*, 2020.

[21] J. C. Hoe, D. Burger, J. Emer, D. Chiou, R. Sendag, and J. Yi, "The future of architectural simulation," *IEEE Micro*, vol. 30, 2010.

[22] A. Robertson and R. Ibbett, "Hase: a flexible high performance architecture simulator," in *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 1, 1994.

[23] Y. Jung, Y. Chiba, D. Kim, and Y. Kim, "simcore: an event-driven simulation framework for performance evaluation of computer systems," in *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728)*, 2000.

[24] C. Collange, M. Daumas, D. Defour, and D. Parello, "Barra: A parallel functional simulator for gpgpu," in *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2010.

[25] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, 2002.

[26] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, 2002.

[27] "The arm research starter kit: System modelling using gem5," https://github.com/arm-university/arm-gem5-rsk, Accessed Feb. 2023.

[28] X. Zhan, Y. Bao, C. Bienia, and K. Li, "Parsec3.0: A multicore benchmark suite with network stacks and splash-2x," *SIGARCH Comput. Archit. News*, vol. 44, feb 2017. [Online]. Available: https://doi.org/10.1145/3053277.3053279

[29] "Intel® vtune™ profiler," https://www.intel.com/content/www/us/en/developer/tools/oneapi/vtune-profiler.html#gs.jescps, 2021.

[30] A. Yasin, "A top-down method for performance analysis and counters architecture," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014.

[31] T. Gleixner and I. Molnar, "perf," https://github.com/torvalds/linux, 2008.

[32] "Reading m1 performance counters," https://gist.github.com/ibireme, commit = 173517c208c7dc333ba962c1f0d67d12, Accessed Nov. 2022.

[33] "Apple mac m1 microarchitectural features," https://https://everymac.com/, Accessed Dec. 2022.

[34] SPEC, "Spec 2017 documentation," https://www.spec.org/cpu2017/Docs, year = 2017,.

[35] R. Panda, S. Song, J. Dean, and L. K. John, "Wait of a decade: Did spec cpu 2017 broaden the performance horizon?" in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[36] A. Limaye and T. Adegbija, "A workload characterization of the spec cpu2017 benchmark suite," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2018.

[37] N. Pemberton and A. Amid, "Firemarshal: Making hw/sw co-design reproducible and reliable," in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021.

[38] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, 2020.

[39] S. Kanev, J. P. Darago, K. Hazelwood, P. Ranganathan, T. Moseley, G.-Y. Wei, and D. Brooks, "Profiling a warehouse-scale computer," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2749469.2750392

[40] W. H. Wang, J.-L. Baer, and H. M. Levy, "Organization and performance of a two-level virtual-real cache hierarchy," *SIGARCH Comput. Archit. News*, vol. 17, apr 1989. [Online]. Available: https://doi.org/10.1145/74926.74942

[41] A. Arcangeli, "Transparent hugepage support," in *KVM forum*, vol. 9, 2010.

[42] "Intel optimizations for dynamic language runtimes," https://github.com/intel/iodlr, Accessed Oct. 2022.

[43] "Runtime performance optimization blueprint: Intel architecture optimization with large code pages," https://www.intel.com/content/www/us/en/developer/articles/technical/runtime-performance-optimization-blueprint-intel-architecture-optimization-with-large-code.html, Accessed Dec. 2022.

[44] "libhugetlbfs," https://github.com/libhugetlbfs/libhugetlbfs, Accessed Dec. 2022.

[45] Z. Tan, A. Waterman, H. Cook, S. Bird, K. Asanović, and D. Patterson, "A case for fame: Fpga architecture model execution," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1815961.1815999

[46] J. Chen, L. K. Dabbiru, D. Wong, M. Annavaram, and M. Dubois, "Adaptive and speculative slack simulations of cmps on cmps," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010.

[47] A. F. Rodrigues, G. R. Voskuilen, S. D. Hammond, and K. S. Hemmert, "Structural simulation toolkit (sst)." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2016.

[48] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "Performance characterization of in-memory data analytics on a modern cloud server," in *2015 IEEE Fifth International Conference on Big Data and Cloud Computing*, 2015.

[49] A. Yasin, Y. Ben-Asher, and A. Mendelson, "Deep-dive analysis of the data analytics workload in cloudsuite," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014.

[50] J. Haj-Yihia, A. Yasin, Y. B. Asher, and A. Mendelson, "Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems," *ACM Trans. Archit. Code Optim.*, vol. 13, dec 2016. [Online]. Available: https://doi.org/10.1145/3018112

[51] G. Ayers, J. H. Ahn, C. Kozyrakis, and P. Ranganathan, "Memory hierarchy for web search," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[52] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson, "Tas: Tcp acceleration as an os service," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303985

[53] W. Gao, J. Zhan, L. Wang, C. Luo, D. Zheng, F. Tang, B. Xie, C. Zheng, X. Wen, X. He, H. Ye, and R. Ren, "Data motifs: A lens towards fully understanding big data and ai workloads," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3243176.3243190

[54] Z. Jia, J. Zhan, L. Wang, C. Luo, W. Gao, Y. Jin, R. Han, and L. Zhang, "Understanding big data analytics workloads on modern processors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, 2017.

[55] A. J. Awan, M. Brorsson, V. Vlassov, and E. Ayguade, "How data volume affects spark based data analytics on a scale-up server," in *BPOE*. Springer, 2015.

[56] A. Lottarini, A. Ramirez, J. Coburn, M. A. Kim, P. Ranganathan, D. Stodolsky, and M. Wachsler, *Vbench: Benchmarking Video Transcoding in the Cloud*. New York, NY, USA: Association for Computing Machinery, 2018, p. 797–809. [Online]. Available: https://doi.org/10.1145/3173162.3173207

[57] A. Addisie, H. Kassa, O. Matthews, and V. Bertacco, "Heterogeneous memory subsystem for natural graph analytics," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018.

[58] J. Takemasa, Y. Koizumi, and T. Hasegawa, "Toward an ideal ndn router on a commercial off-the-shelf computer," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ser. ICN '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3125719.3125731