Sources of Error in Full-System Simulation

Anthony Gutierrez Joseph Pusdesris
Ronald G. Dreslinski Trevor Mudge
Advanced Computer Architecture Laboratory
University of Michigan - Ann Arbor, MI
{atgutier, joemp, rdreslin, tnm}@umich.edu

Chander Sudanthi Christopher D. Emmons
Mitchell Hayenga Nigel Paver
ARM Research - Austin, TX
{chander.sudanthi, chris.emmons, mitch.hayenga, nigel.paver}@arm.com

***Abstract*—In this work we investigate the sources of error in gem5—a state-of-the-art computer simulator—by validating it against a real hardware platform: the ARM Versatile Express TC2 development board. We design a custom gem5 configuration and make several changes to the simulator itself in order to more closely match the Versatile Express TC2 board. With the modifications we make to the simulator, we are able to achieve a mean percentage runtime error of 5% and a mean absolute percentage runtime error of 13% for the SPEC CPU2006 benchmarks. For the PARSEC benchmarks we achieve a mean percentage runtime error of -11% and -12% for single and dual-core runs respectively, and a mean absolute percentage runtime error of 16% and 17% for single and dual-core runs respectively. While prior work typically considers only runtime accuracy, we extend our investigation to include several key microarchitectural statistics as well, showing that we can achieve accuracy within 20% on average for a majority of them. Much of this error is likely from modeling similar, but not identical components.**

***Keywords*—computer architecture, computer simulation**

I. Introduction

System-on-chip (SoC) computing systems are complex and highly integrated—comprising CPUs, caches, I/O and memory controllers, GPUs, and accelerators on a single chip. Prototyping these SoC devices is exceedingly expensive, therefore computer architects rely on simulation to evaluate new ideas in a timely and cost-effective manner. However, there is a tradeoff between simulator performance, accuracy, and flexibility that inherently leads to some amount of experimental error. This error, while expected, is often not well-understood by users. If conclusions are to be drawn from simulation results it is important to understand the sources of simulation error, as well as how they impact overall simulation accuracy—typically taken to be benchmark runtime. Unfortunately, accounting for simulation error is difficult because of the increased complexity of hardware systems and the simulators that model them. This makes it difficult for small research teams to validate and maintain such complex simulators, and has resulted in a widespread use of open-source projects that may not be validated or are validated only for a specific purpose.

Black and Shen classify the sources of simulation error into three separate categories: 1) *modeling errors* occur when the simulator developer codes the desired functionality erroneously; 2) *specification errors* occur when the developer is uninformed or must speculate about the behavior of the functionality being modeled; 3) *abstraction errors* occur when the developer abstracts away, or fails to implement, certain details of the system being modeled [4]. If a simulator is widely-used and well-maintained, modeling errors will be gradually corrected. Specification and abstraction errors, however, may remain in the simulator for several reasons—certain features of a system may not be publicly available, or the abstractions may simplify the simulator code and improve simulation speed. These abstractions may not impact the accuracy of the simulator significantly for its intended usage scenario, however they may cause experimental error that is unacceptable when the simulator is modified to represent a system it was not originally intended to model. Therefore, it is necessary to validate the simulator to ensure that these errors do not lead to unreliable results.

Simulation technology has advanced significantly in recent years. Several modern simulators are capable of modeling multiple ISAs, several CPU types, various interconnect topologies, and devices with enough detail that they are able to boot unmodified operating systems, and in some cases run interactive graphical workloads. The addition of such sophisticated capabilities has increased the number of specification and abstraction errors in modern simulators. One obvious abstraction error is the lack of a GPU model in most architectural simulators. As a result they are not able to accurately simulate systems running emerging and important classes of workloads, such as interactive graphical workloads typified by *BBench* [11]. When run on gem5, BBench exhibits a runtime error of up to 3× because it lacks a GPU model, which forces the system to use software rendering.

While several past works have validated various simulators [4, 6, 8, 10, 23], they focus on a single metric, such as runtime or memory latency. They do not consider the accuracy of microarchitectural characteristics, nor do they consider how microarchitectural accuracy affects overall runtime accuracy. Even though runtime performance remains an important metric, many papers in the architecture field focus on improving other microarchitectural characteristics—e.g., reducing cache or TLB accesses to save energy—in spite of the fact the simulator has only been validated for runtime. It is important to revisit simulator validation for a current state-of-the-art simulator to ensure that its advanced features are representative of today's complex systems and that it is capable of capturing the behavior of emerging workloads.

One example of a sophisticated modern simulator is the *gem5* simulator [2], which is the subject of this work. gem5 is a widely used simulator that is capable of modeling several ISAs, including the two most popular ISAs in use today: ARM and x86. It does so with enough detail that it is capable of booting unmodified versions of popular Linux distributions, such as Ubuntu. In the case of ARM, gem5 is also able to boot the latest version of Google's Android operating system. In this

Simulator	Maintained	ISA(s) Supported	License	Accuracy	Guest OS(s)
Flexus	No	SPARC, x86 (Requires Simics)	BSD	Cycle-Accurate	Linux, Solaris
gem5	Yes	Alpha, ARM, MIPS, POWER, SPARC, x86	BSD	Cycle-Accurate	Android, FreeBSD, Linux, Solaris
GEMS	No	SPARC (Requires Simics)	GPL	Timing	Solaris
MARSS	Yes	x86 (Requires QEMU)	GPL	Cycle-Accurate	Linux
OVPsim	Yes	ARM, MIPS, x86	Dual	Functional	Android, Linux, and others
PTLsim	No	x86 (Requires Xen and KVM/QEMU)	GPL	Cycle-Accurate	Linux
Simics	Yes	Alpha, ARM, M68k, MIPS, POWER, SPARC, x86	Closed	Functional	FreeBSD, Linux, NetBSD, Solaris, Windows, and others

TABLE I: A taxonomy of modern full-system computer simulators.

work we validate the gem5 simulator against an *ARM Versatile Express TC2 (VExpress)* development board and make the following contributions:

- We show that by modifying the gem5 simulator, and configuring it properly, we are able to reduce the mean percentage error (MPE) of runtime to 5% and the mean absolute percentage error (MAPE) of runtime to 13% for the SPEC CPU2006 benchmarks. For the PARSEC benchmarks we are able to reduce the MPE of runtime to -11% and -12% for single and dual-core runs respectively, and the MPAE is reduced to 16% and 17% for single and dual-core runs respectively.
- We evaluate the PARSEC benchmarks, a set of commonly used multi-threaded benchmarks, and show that gem5's multi-threaded speedup has a MPE of 0.43% and a MAPE of 0.72% with respect to the VExpress' multi-core speedup.
- We measure the accuracy of several key microarchitectural statistics and show that, on average, most are accurate to within 20%.
- We give examples of specification errors present in gem5 and show how we addressed them, thereby improving the accuracy of some key microarchitectural statistics.

The rest of this paper is organized as follows: in section II we describe modern full-system simulators and the criteria we used to select which simulator to focus on for this work. Section III describes the architecture of the VExpress board in detail and how we configured gem5 to match it. We describe our experimental methodology in section IV. Our results are presented in section V. Related work is discussed in section VI. We reflect and discuss the results in section VII and conclude in section VIII.

II. Computer Simulators

Due to the high cost of hardware prototyping and the low performance of gate-level simulation, computer simulators have been relied upon by the architecture community for many years. SimpleScalar [5] was one of the earliest architectural simulators to achieve widespread use, primarily because of its accuracy, performance, and usability. Soon, the simulation landscape expanded and there were a variety of simulators for nearly any use-case. SimOS [22] was one of the first successful attempts at full-system simulation. SimOS pioneered the use of techniques such as direct-execution and dynamic binary translation to provide fast simulation. As hardware performance increased, computer simulators were able to model real systems with a greater level of detail without sacrificing performance. Today there are a large number of full-system simulators available for a variety of purposes.

When choosing which modern simulator would be most suitable for validation, we considered several criterion such as: support for relevant ISAs, level of modeling accuracy, flexibility, licensing, available workloads, and whether or not the project has active support. Table I lists the most popular full-system simulators in use today and shows how each matches the criteria listed above.

A. Related Full-System Simulators

Flexus [27] is designed for fast and accurate full-system simulation via rigorous statistical sampling; it implements *SMARTS* [28]. By checkpointing and running only the necessary regions with a fully-detailed model, simulator performance is vastly improved, allowing complicated and long-running benchmarks to be simulated.

GEMS [18] was primarily designed to model the memory system—caches, interconnect, coherence protocols, and topology—with a high-level of flexibility. It features *SLICC*, a domain-specific language for specifying custom coherence protocols. GEMS requires the use of Simics, thus decoupling functional execution from its timing models.

MARSS [21] and *PTLsim* [29] were designed to provide fast simulation of x86 systems by integrating closely with a hypervisor/virtual machine or emulator. By using co-simulation a virtual machine may be switched seamlessly between the host system's CPU and the simulated CPU, allowing the region(s) of interest to be simulated in full-detail while all others can be forwarded through at near real-time speeds.

OVPsim [13] and *Simics* [17] are designed to provide complete modelling of all necessary components to run a variety of unmodified operating systems, as well as very fast functional simulation. However, they require the use of additional simulation technologies to model the systems with higher levels of detail.

B. The gem5 Simulator

The *gem5* simulator [2] is a collaborative project based on the *M5* simulator [3] and the *Ruby* component of GEMS.

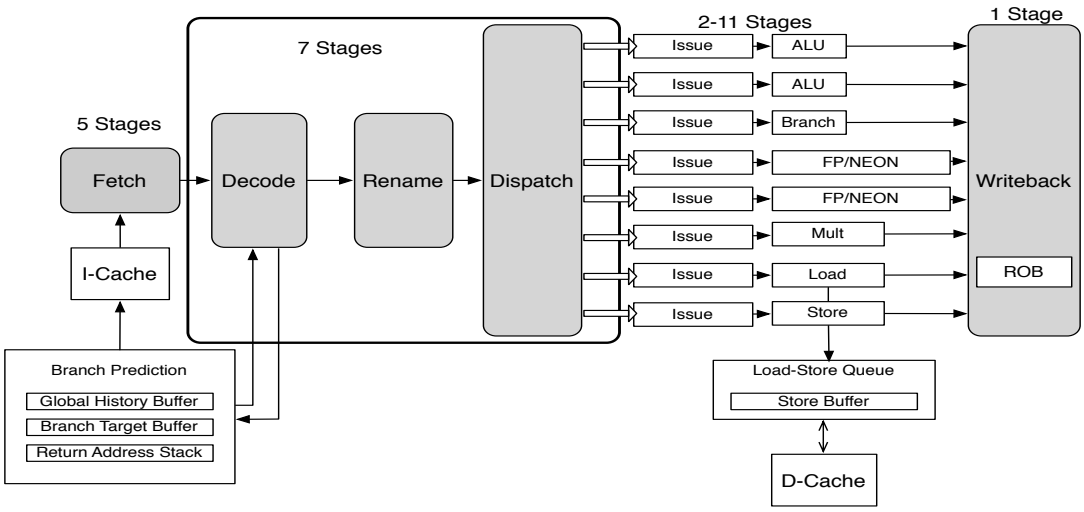


Fig. 1: The Cortex-A15 pipeline.

The M5 simulator is a full-system simulator designed to model networked systems and Ruby is the memory system component of GEMS. gem5 provides detailed models for both in-order and out-of-order (OoO) CPUs, as well as simple models for fast functional simulation. Multi-threading is supported via both CMP-style systems, and SMT-enabled CPUs. It provides two modes: *system call emulation* mode and *full-system* mode. In system call emulation mode binaries run directly on the simulator and all operating system functionality is emulated. The memory system allows users to define a variety of cache organizations, e.g., directory-based or bus-based, and coherence protocols (it incorporates SLICC). We choose to evaluate the gem5 simulator in this study for a variety of reasons:

- It supports the two most popular ISAs in use today: ARM and x86, among others.
- It is widely-used in academia and industry, and has a BSD-style license; as such it is important to inform the community of its accuracy.
- It is able to boot unmodified versions of several relevant operating systems, e.g., Google's Android OS, and Ubuntu Linux, and it can run interactive workloads.
- It has incorporated most, if not all, of the advanced features of the other full-system simulators: system networking, KVM [14] integration, checkpointing support, simpoint [24] generation, detailed DRAM models, and a configurable cache system.

III. Modelling the Versatile Express TC2 Platform

We choose an ARM Versatile Express (VExpress) platform because it is the baseline platform on which the ARM implementation of gem5 is based, and because of the sophistication of the CPU models for the ARM ISA in gem5. In order to properly configure gem5 to model the VExpress board we identify its key properties and the corresponding models within gem5.

The VExpress board utilizes a *CoreTile Express* SoC, which contains two Cortex-A15 processors, three Cortex-A7 processors, two layers of cache memory, I/O, and memory controllers. We focus on the details of its CPU features, memory system, and interconnect when configuring gem5. Because gem5 does not currently have an in-order CPU model for the ARM ISA, we disable the A7 on the VExpress board and only use the A15 CPU. Details of the A15's microarchitecture are inferred from: [7, 15, 25].

A. ARM Cortex-A15 Microarchitecture

A high-level diagram of the A15 pipeline is shown in figure 1. The A15 is an OoO, 3-issue pipeline, with between 15 and 24 stages for integer and floating point/NEON instructions respectively; loads and stores require 18 stages. The A15 is able to fetch, decode, and issue up to three instructions to its eight execution units every cycle.

1) Instruction Fetch: The A15 contains a 32kB, physically-indexed, physically-tagged, 2-way set-associative L1 instruction cache with 64B lines. The instruction cache supports unaligned accesses and can supply up to 16 bytes per fetch, which are stored in a fetch buffer that may service subsequent accesses to the same 16 byte region. Because fetch reads at most 16 of the 64 bytes in the cache line, only the critical words of the line are allocated to the fetch stage. The instruction cache receives data from the L2 cache via two fill buffers, which are non-blocking—an instruction hit can proceed past a miss before its data have come back from the L2 cache.

The fetch stage contains the branch prediction unit. The branch prediction unit consists of a two-level global history predictor with a taken array, a not taken array, and a choice predictor. The separation of taken and not taken arrays is similar to the bi-mode branch predictor [16]. Each of the global history arrays has 8K entries and contain the status predictors. A 64 entry, fully-associative micro-BTB caches taken branch targets for direct branches—this removes the need for an extra pipeline bubble for each predicted branch. The main BTB is 2K entries and may override the micro-BTB's prediction. For

indirect branches, a 256 entry BTB indexed by the XOR of the branch history and the PC is used to predict the branch target. The direction of indirect branches are predicted with a separate indirect predictor. Branches are resolved, and may flush the pipeline on a misprediction in the execute stage. Incorrect branch target prediction may be detected in the decode stage.

2) Decode, Rename, and Dispatch: Up to three instructions may be decoded per cycle. The decode stage also contains a 32 entry loop buffer that can store up to two forward branches and one backwards branch. When a loop is detected by the decode stage, the fetch stage is completely disabled, as are most of the decode stages. This can significantly reduce the energy consumption of the processor front-end.

The rename stage maps the A15's 16 architected registers to its 128-entry physical register file. Two separate rename tables are maintained—one for the ARM registers and one for the NEON registers. While integer and floating point/NEON instructions have separate architected register files, they share the same set of physical registers. The result queue holds speculative register results that are awaiting writeback to the register file.

Dispatch takes up a single stage and sends up to three instructions to the A15's eight functional units.

3) Execute: The A15 has 8 separate functional units that are clustered by type. As figure 1 shows, there are two single-stage integer pipelines for ALU and shift operations, one four-stage multiplier, two floating point/NEON units that may be up to 10 stages, and two four-stage units for load/store operations. All execution pipelines are fixed-length with the exception of the floating point/NEON engine, which may take 2-10 cycles; loads take 4 cycles, but may stall in the case of a data cache miss.

The floating point/NEON units are not identical—only one of the two is capable of executing 128 bit operations, e.g., the quad floating point multiply-accumulate operations. Similarly, the two load/store pipelines are not identical—one is exclusively for loads, the other exclusively for stores, however they cooperate with each other to resolve data dependences. Loads may be issued out-of-order, but they may not be ordered before stores to the same address. The four stages of the load/store units are responsible for address generation and TLB lookup, address setup to tag and data arrays, data/tag access, and data selection/formatting/forwarding.

B. Memory System

The memory system of the A15 consists of two levels of on-chip cache—split, private L1 instruction and data caches, and a shared L2 cache. Table II shows the relevant cache parameters for the VExpress. The data caches are non-blocking and can hold up to six requests in their MSHRs; loads or stores to different addresses can hit in the cache and bypass these requests. Loads may be speculatively executed, but speculative stores are not committed to memory. Loads may forward data from previous, speculatively-executed stores. The L1 caches are physically-indexed, physically-tagged and use a true LRU replacement policy.

The L2 cache strictly enforces inclusion, i.e., any line contained in any of the system's L1 caches must also reside in the L2 cache. A hardware prefetcher is included with the L2 cache. It can detect and prefetch data load/store streams via a stride pattern; for instruction fetches the prefetcher uses a next-line mechanism. At most six prefetch requests may be outstanding and the prefetcher may not prefetch across page boundaries. The L2 cache is physically-indexed, physically-tagged and uses a random replacement policy. Coherence between the L2 cache and the L1 cache is maintained using a hybrid MESI/MOESI protocol; the L1 data cache uses MESI and the L2 cache uses MOESI.

. The VExpress board has 1GB of DDR2-800 memory with a x32 interface clocked at 400MHz (CL=5, tRCD=5, tRP=5). There are two devices—each device is 4Gb (the TwinDie DRAM from Micron comprises two 2Gb Micron DDR2 dies), with eight banks per rank in a 32Mx8 configuration, and two ranks per device [26]. The system bus is clocked at 500MHz.

C. Configuring gem5

Because each component modeled in gem5 is parameterized, we have a high degree of flexibility when configuring it to match the VExpress board. We use publicly available documentation to set the parameters of each object. For others we tune their configurations based on empirical observations. For some components configuration is not enough, i.e., the implementation matters and we must provide new implementations, or alter the existing implementations for these components. Furthermore, some features of the A15 would require significant effort to model, therefore we disable them on the VExpress board where possible; see the descriptions of the *Auxiliary Control Register* and the *L2 Prefetch Control Register* in the A15 TRM [7] for information on how we disabled certain features.

1) O3 CPU Model: gem5's *O3 CPU* models a 7-stage pipelined OoO processor, and each stage itself has configurable depth and width parameters. The O3 CPU model also includes components such as the branch predictor, TLBs, load/store queue, etc. Because the default ARM configuration in gem5 already has pipeline parameters that are similar to the A15, we do not make any changes to the pipeline parameters. The default configuration also has the correct number and type of functional units.

There are however, several changes that we had to make to the configuration of other components, as well as to the O3 CPU code itself. The instruction and data TLBs are both set to 64 entries by default. The A15 however, has a 32-entry instruction TLB, and two separate 32-entry data TLBs (one for reads and one for writes). We configure the I-TLB to have 32 entries and we leave the D-TLB at 64 because its aggregate capacity is 64 in the real hardware.

gem5 implements a two-level tournament branch predictor, which is not entirely representative of the bi-mode predictor in the A15. We implement the bi-mode branch predictor in gem5, and set the sizes of its structures in accordance with the A15. Because gem5 has a single branch predictor that services all branches, we disable the A15's indirect predictor. gem5 also seems to over serialize branch execution, to account for this we force in-order issue to the branch execution unit in the A15.

The cache line returned from a fetch is stored in a fetch buffer, which is the size of an entire cache line in gem5. The fetched data is kept in the fetch buffer and is used to service subsequent fetches. Because the A15's fetch buffer is only 16B, gem5 drastically underestimated the number of instruction cache accesses. To correct this we modified the

Parameter	VExpress	Validated gem5
Pipeline	ARM Cortex-A15	O3 Model
CPU clock	1GHz	1GHz
Branch predictor	Bi-Mode	Bi-Mode
Fetch buffer size	16B	16B
Cache block size	64B	64B
L1 I-cache size	32kB	32kB
L1 I-cache associativity	2-way	2-way
L1 D-cache size	32kB	32kB
L1 D-cache associativity	2-way	2-way
L2 cache size	1MB	1MB
L2 cache associativity	16-way	16-way
DRAM	DDR2 x32	DDR2 x32
DRAM frequency	400MHz	400MHz
Memory size	1GB	1GB
Bus width	128b	128b
System bus frequency	500MHz	500MHz

TABLE II: Parameters for the VExpress board and our gem5 configuration.

fetch buffer to ensure that it only held the critical 16B returned after a fetch, which more closely matches the A15.

The A15's fetch stage is able to fetch instruction across cache line boundaries, i.e., if a 16B fetch buffer segment straddles two cache lines, the A15 will fetch both cache lines, extract the critical 16B, and place them into the fetch buffer. gem5's default fetch stage always aligns fetches to cache line boundaries. To address this we modified the fetch stage in gem5 to support misaligned fetching of the critical 16B of a fetch.

The default clock in gem5 is 1GHz for ARM, which is the clock frequency of the A15 on the VExpress board.

2) Memory System: gem5's memory system uses the MOESI coherence protocol and is a good approximation of the A15's coherence protocol. gem5 provides private, split L1 data and instruction caches. Several additional layers of caches may be added to the cache hierarchy, and may private or shared among any of the cores in the system.

The default size of the data cache is 64kB; we manually set it to 32kB to match the A15's data cache size. Several of the L2 caches parameters are modified—the 2MB default L2 cache size in gem5 is too large, therefore we set it to 1MB matching the VExpress board's SoC. The associativity is changed from 8 to 16. gem5's caches use an LRU replacement policy, however the A15 has a random L2 replacement policy. To address this we implemented a psuedo-random replacement policy for the L2 cache in gem5.

By default the instruction caches in gem5 ignore any I-cache invalidation instructions, which could potentially skew the I-cache miss statistics. We observed frequent *ICIALLUIS* instructions during the execution of the SPEC and PARSEC benchmarks. The ICIALLUIS instruction invalidates all entries in the I-cache, and is mostly used by the kernel to invalidate cache contents after context switches, which are assumed to be cold after switching. To address this, we implemented the proper invalidation behavior in the caches when the ICIALLUIS instruction is encountered.

We use the *SimpleDRAM* model in gem5, configuring it to match the DDR2-800 x32 DRAM configuration of the VExpress board. The parameters used in the SimpleDRAM model are derived from the data sheet of the onboard Micron memory as mentioned in section III-B. The VExpress board contains static latencies—e.g. bus and bridge latencies—that we do not have visibility of. To take into account this unknown latency, we empirically determine and add 20ns of static latency to the gem5 DRAM configuration.

Finally, due to differences in the gem5 prefetcher and the prefetcher in the A15 we disable all prefetching in both gem5 and the VExpress board. The A15 also has special hardware that detects and accelerates streaming memory accesses—e.g., a call to *memcpy*—we disable all hardware streaming acceleration.

IV. Experimental Methodology

All experiments are run on gem5 and the VExpress board. The gem5 configuration parameters, shown in table II, are set to match those of the VExpress board as closely as possible. Beyond configuring gem5 to match the parameters that are publicly available, the memory latency is empirically tuned.

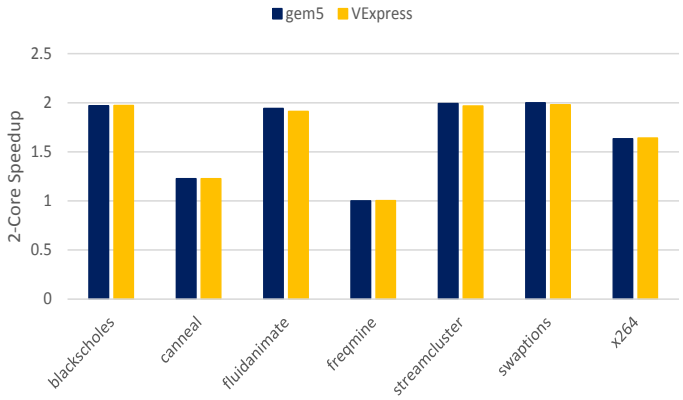
A. Benchmarks and Runtime Environment

The SPEC CPU2006 benchmarks [12] were chosen, because in addition to being the accepted standard, they are commonly used in publications employing gem5. Due to what introspection is possible on the VExpress board, benchmarks must run to completion in order for us to acquire the data necessary for this study. As a result of gem5's slowdown relative to real hardware, we use a subset (11 out of 29) of the SPEC benchmarks, using the *train* input set. These are the benchmarks that are able to run to completion on gem5's detailed OoO model in a reasonable amount of time, and that have very low variation in performance statistics between runs on the VExpress. All SPEC benchmarks were statically compiled using the Linaro *arm-linux-gnueabi* toolchain version 4.6.3-1ubuntu5, which allowed the same binary to be run on the VExpress and gem5.

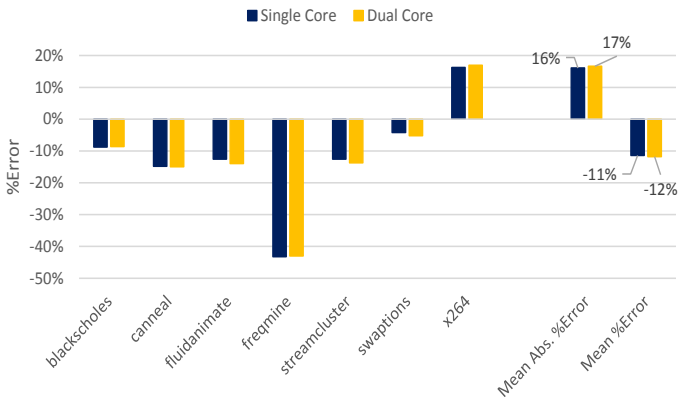
To evaluate the accuracy of multi-core scaling on gem5 we use the PARSEC benchmarks [1], which are widely-accepted as the standard multi-threaded benchmarks. The PARSEC benchmarks were compiled with the same compiler used to compile the SPEC benchmarks. Due to simulation time constraints we run 7 out of the 13 PARSEC workloads and we use the *simsmall* input set. All benchmarks were compiled statically and the same binary was run on both gem5 and the VExpress board.

Both gem5 and the VExpress run stripped-down versions of Linux—a headless version of Ubuntu 13.04 from Linaro on the VExpress, and ARM Embedded Linux Filesystem 5.0.0 Jakarta on gem5 (available on the gem5 wiki [9])—with all background tasks and unneeded processes halted. To reduce variation caused by the differences between the VExpress' Flash disk and gem5's disk model, we ran all workloads from a RAM based filesystem—*tmpfs*.

Perf was used to collect performance counter information on both hardware and in gem5. To support perf on gem5 we implemented the performance counters described in the architecture reference manual. Our implementation works by relaying information from gem5 statistics to the guest software when the associated register is queried. This was crucial to precisely replicating the methodology between the two systems. Benchmarks are run 10 times on hardware, and averaged to



(a) PARSEC runtime scaling accuracy.



(b) PARSEC runtime accuracy.

Fig. 2: Runtime and scaling accuracy of PARSEC. As can be seen in figure 2b, the runtime MAPE and MPE is consistent across 1 and 2 core runs; because of this, the scaling error is less than 1%, which is shown in figure 2a.

establish a stable reference point against which to compare gem5. In order to further reduce variance, one of the two A15 cores, and all three of the A7 cores, on the VExpress board were powered off to eliminate migration, unusual scheduling, and to limit the counter polling to only one core. While we use Perf to measure all relevant microarchitectural statistics, we do not report the number of instructions because they are always accurate to within a fraction of a percent. Because of this accuracy we use the total count for each statistic to measure error, as opposed to misses or accesses per thousand instructions (M/APKI), however we report MPKI and APKI for the relevant statistics.

We use STREAM [19] and LMbench [20] to evaluate the memory system accuracy. STREAM is a synthetic memory benchmark that measures sustainable memory bandwidth, and is designed to work with datasets larger than the cache capacity. LMBench is a suite of benchmarks for various I/O operations, however we use it only to measure memory latency.

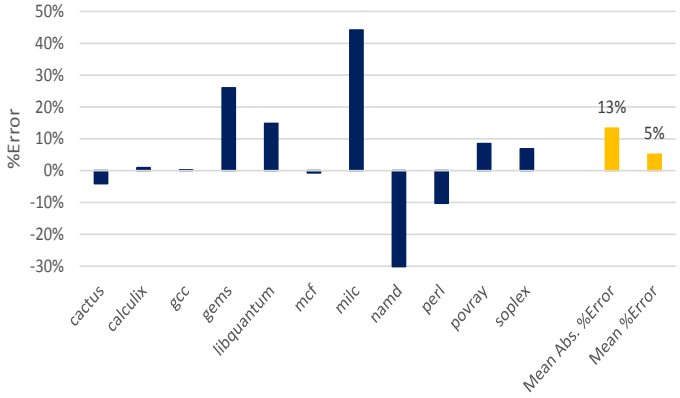


Fig. 3: Runtime accuracy of SPEC. The MAPE and MPE for the SPEC benchmarks are 13% and 5% respectively. All benchmarks are within 30% on average, with the lone exception being milc, which exhibits much worse branch miss rates on gem5.

V. Results

Due to space constraints we only show the statistics for our final gem5 configuration, and for PARSEC we only show multi-core scaling and runtime accuracy. However, to make the raw data from all of our experiments public, including the accuracy improvement provided by each individual change we made to the simulator, as well as all detailed analysis of the accuracy of the microarchitectural statistics for PARSEC. This data will be available at: http://web.eecs.umich.edu/~atgutier/gem5.html.

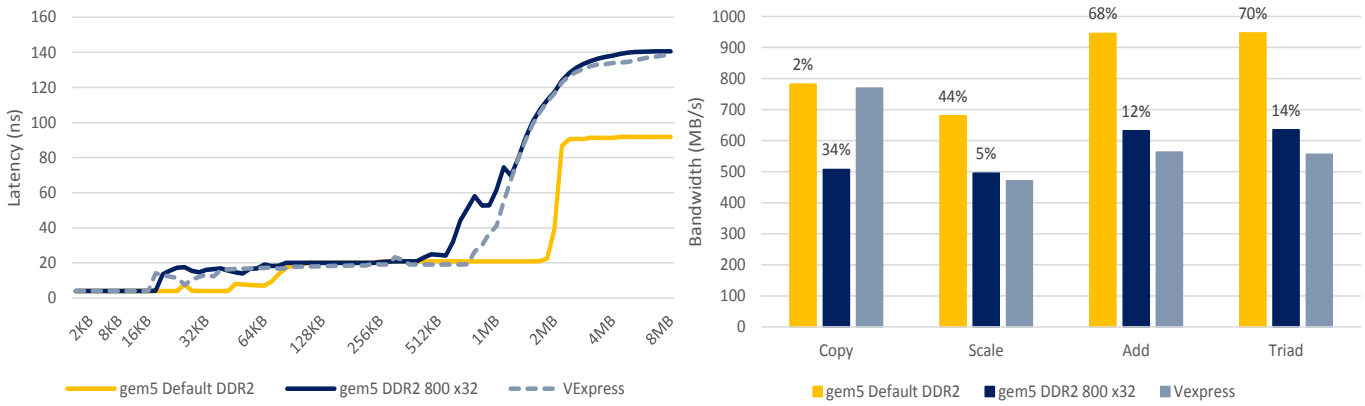
A. Runtime Accuracy

With our configuration, we achieve a MAPE for the SPEC benchmarks of only 13%, and most benchmarks are within 10%. The three most significant outliers are SPECFP benchmarks. See figure 3 for further explanation. For PARSEC we achieve a MAPE of 16% and 17% for single and dual-core runs respectively, see figure 2b. It is important to note that our approximation achieves this level of accuracy without modeling the microarchitecture precisely.

Of the three sources of simulation error described earlier, specification error is the dominant source in this work. Due to the activity of the gem5 developer community, significant modeling errors are assumed to have been worked out. For target workloads, abstraction error is insignificant because the main devices that are exercised on the bus are components that are modeled well in gem5—the CPU and memory. Though, relatively comparing SPECINT and SPECFP benchmarks, there is more abstraction error in SPECFP. The gem5 community has time less time constructing an accurate floating point pipeline implementation, that it has on the other execution pipelines. Because modeling and abstraction error are insignificant, specification error is the main source of error in our results.

B. Multithreaded Scaling

In addition to runtime accuracy we also measure the accuracy of multithreaded scaling in gem5. To do this we



(a) Memory latency. **(b) Memory bandwidth.**

Fig. 4: Observed memory access latency and bandwidth. The latency for each level of memory, as well as gem5's DDR2 controller, must be tuned as a pre-requisite for other statistics to correlate. Figure 4a shows the result of this tuning. The cache sizes, 32k and 1M, create visible steps in access latency. Figure 4b shows the difference in memory bandwidth between the configurations. The values above the bars represent the absolute percentage error with respect to the VExpress.

compare the speedup of the PARSEC benchmarks on two cores over a single-threaded version. We do not measure beyond dual-core scaling because the VExpress contains only two A15 cores. Figure 2a shows the result of this experiment. The scaling demonstrated by gem5 matches the VExpress to within 1% for all benchmarks.

C. Memory System Accuracy

Figure 4 shows the measured latency and bandwidth for both gem5 and the VExpress board. The memory latency for each level of the memory hierarchy is shown in figure 4a. In the default gem5 configuration there is a steep increase in latency when transitioning from the L2 cache to memory. This steep increase is caused by an L2 replacement policy specification error. The default LRU replacement policy in gem5 is changed to a random replacement policy. In addition, the DDR controller in gem5 is configured to match the VExpress board as described in section III-B. With the replacement policy and DRAM controller configuration fixes, we are able to closely match the latency curve of the VExpress board.

The bandwidth, shown in figure 4b, correlates to within 5%-15% for all components of STREAM, with the exception of copy. It is currently unclear why this is the case, given that all prefetching and hardware streaming support was disabled on the VExpress. The default memory controller configuration consistently over estimates bandwidth, with error from 44% to 70% for all components of STREAM except copy.

D. Microarchitectural Accuracy

In addition to overall performance, we evaluate the accuracy of several key microarchitectural statistics. Figures 5, 6, and 7 show the percent difference for the SPEC benchmarks relative to the VExpress board, for each of the microarchitectural statistics we measured. These figures also report M/APKI, and the benchmarks are sorted by M/APKI from lowest to highest. Note that the scale for the M/APKI is logarithmic. As seen in this graph, each of the microarchitectural statistics is within 35% on average for the validated configuration, with the exceptions being TLB misses. The D-TLB error is likely due to the fact that gem5 utilizes a single 64-entry TLB, whereas the A15 has separate 32-entry TLBs for reads and writes respectively. If accesses are biased towards either reads or writes the performance will be improved with a single 64-entry TLB because all 64 entries are available to any access. For the I-TLB the MPKI is so low that slight variation can cause large error, however as the I-TLB MPKI increases so does gem5's accuracy.

E. Sources of Specification and Abstraction Errors

We addressed several of the key specification and abstraction errors in our efforts to improve the accuracy of gem5 with respect to the VExpress. Some of the changes were trivial, e.g., setting the proper sizes and latencies for caches, while others required significant modifications to the simulator source itself. We disabled features in hardware when we determined the feature would not be critical to add to gem5, or would take significant effort to add. The indirect branch predictor, out-of-order branch issue, all prefetching, and the loop buffer are all disabled.

Beyond configuring the parameters of the simulator we implemented several key features that we believed were important for accuracy, these include: the bi-mode branch predictor, a configurable fetch buffer, misaligned fetch, a psuedo-random L2 replacement policy, I-cache invalidation instructions, and the ARM PMU counters so we could use the same measurement tool. We chose to put effort into implementing these particular features by first measuring the error in the simulator as it currently exists to discern which statistics had the largest error, then identifying the components that most contributed to the error. While these changes required significant effort, both in terms of understanding our baseline platform and the simulator source code, they provided greatly improved accuracy for both runtime and microarchitectural statistics.

There are remaining sources of specification and abstraction error in the simulator, particularly in the TLB models and

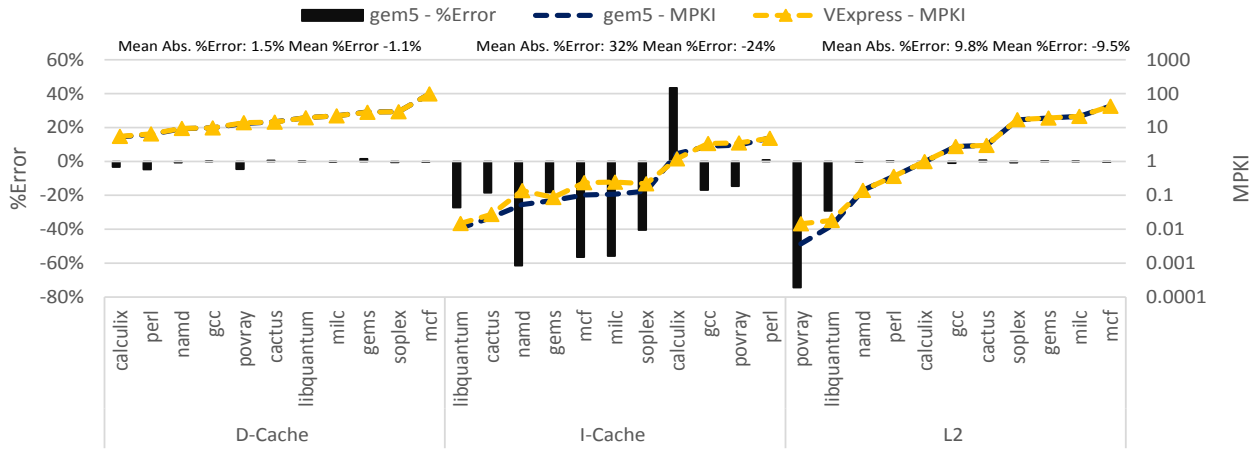


Fig. 5: Cache miss stats for SPEC. The cache models in gem5 accurately model the behavior of the caches in the A15 when they see the same access pattern, which is the case for the D-cache. The I-cache miss statistics, however, are typically underreported on gem5. This error is likely due to the more aggressive fetch stage in the A15, which consists of five stages. The L2 error is likely due to seeing fewer accesses from the I-cache, and from fewer page table walks caused by TLB misses. As the MPKI increases, however, the accuracy of the L2 misses improves—slightly fewer misses can increase error more significantly when the MPKI is low.

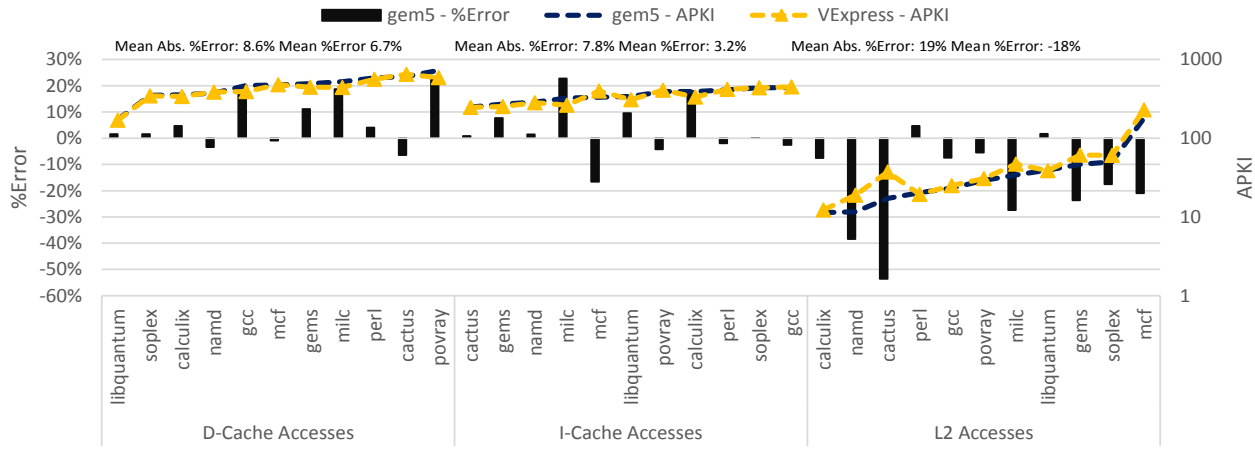


Fig. 6: Cache access stats for SPEC. The cache accesses are accurate to within 10% on average for the D-cache and I-cache. The differences are likely due to a more aggressive fetch stage, and differences in the load-store queue. The error of the L2 cache is around 20% on average, which is due to seeing different accesses coming from the L1 caches and the page table walker.

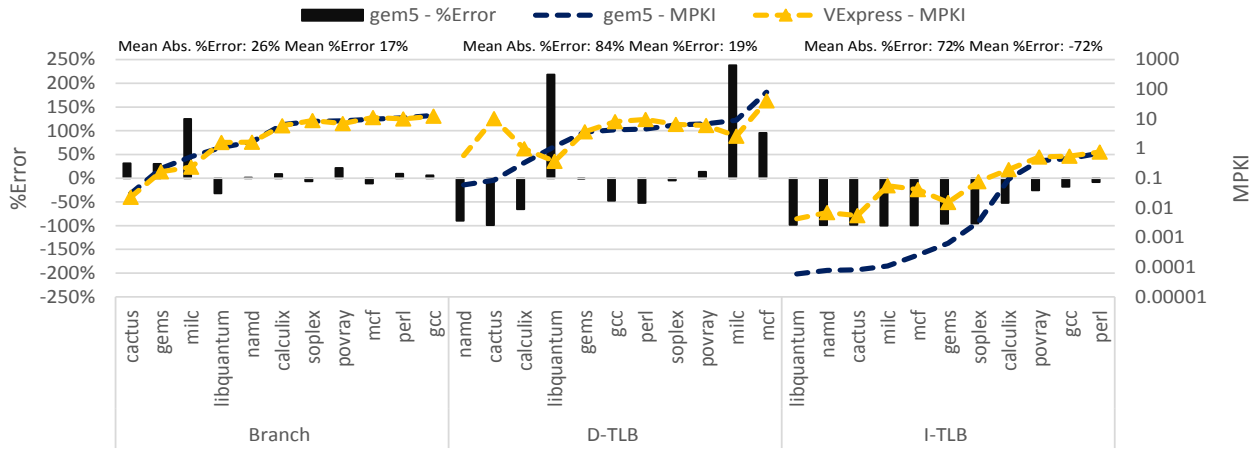


Fig. 7: Branch and TLB miss stats for SPEC. The branch misses are accurate to within 26%, and accuracy improves as the MPKI increases. Because of the extremely low I-TLB miss rates the error is 72% on average, however the accuracy improves as the MPKI increases. The D-TLB misses exhibit large error due to the fact that a single 64-entry TLB is used for loads and stores, which may benefit some workloads, and hurt others.

the front end of the pipeline—as our results showed, I-cache and TLBs had the highest error. gem5's fetch engine only allows a single outstanding I-cache access, whereas modern OoO CPUs are fully pipelined allowing multiple parallel accesses to instruction cache lines. This specification error in the fetch stage contributes to the I-cache miss statistic error. The other remaining source of specification error is in the TLB models. To more accurately model the TLBs in the A15 support, would be needed for separate TLBs for reads and writes, as well as a second level TLB. Inaccurate TLB models also have an effect on L2 accesses as can be seen with the namd and cactus results. The gem5 D-TLB MPKI of namd and cactus results differ the most from the VExpress board. Each D-TLB miss results in a page table walk that makes multiple accesses into the L2. In the VExpress board, there is a L2 TLB to hide the effect of D-TLB misses on the L2 and for namd and cactus there are fewer misses in general than gem5. This has a multiplicative effect on L2 accesses.

VI. Related Work

Desikan et al. [8] validated an Alpha 21264 simulator using a Compaq DS-10L workstation as a baseline. By creating a very detailed model of the Alpha 21264 processor they were able to achieve a 2% CPI error for their microbenchmarks. Despite this, they found that complex interactions in the memory system prevented the formulation of an accurate memory model, and were thus only able to achieve an average CPI error of 18% on SPEC CPU2000 benchmarks.

Gibson et al. [10] provided an analysis of the accuracy of several of the simulators used during the development of the Stanford FLASH multiprocessor. Using the SPLASH-2 benchmark suite they validated simulators of varying complexity against the hardware prototype produced by the project. They found that a simple in-order processor model running at speeds faster than memory, could produce results at least as accurate as an out-of-order model due to failure to accurately capture intricacies of the architecture in the models. Additionally, they found that failure to accurately model aspects of the virtual memory system, such as TLB miss handling and page coloring, can cause significant degradation in the accuracy of a model.

Saidi et al. [23] validated the M5 simulator for network-intensive workloads against a Compaq Alpha XP1000 workstation. Using a series of memory and network microbenchmarks as well as the SPEC WEB99 web server benchmark suite, they found that M5 models often fell within 15% of the hardware. They further observed that TLB miss behavior can cause significant discrepancies between the model and hardware.

Butko et al. [6] analyzed the accuracy of gem5 compared to a Snowball SKYS9500-ULP-C01hardware development kit. Using SPLASH-2, ALPHBench, and STREAM, they found the accuracy of gem5 to vary between 1.39% and 17.94%. However, their analysis compared only benchmark runtimes and L2 miss rates—on the simple functional CPU models—and did not consider contributions to overall error from individual components of the models.

VII. Discussion

While simulators remain necessary research tools, it is important that they are used properly and that any assumptions made about their use are valid. This work offers several reflections on simulator usage:

1) Architectural simulators are not microarchitectural simulators: Full-system simulators are not designed to model each microarchitectural component with perfect accuracy. As such, microarchitectural statistics obtained via simulation should not be relied upon unless they have been validated. For example, the change to gem5's fetch buffer described in section III does not lead to any change in runtime, however it drastically increased the number of instruction cache accesses. If a research idea purported to reduce the number of cache accesses to save power, without being aware of the fetch buffer, its results would be skewed.

2) Specification and abstraction errors do not imply inaccurate simulation: As our results showed, it is possible to maintain accuracy despite the presence of specification and abstraction errors. These errors are acceptable and help to ensure that simulator performance is not a hindrance to evaluating research ideas. As long as the components most necessary for capturing workload behavior are present the simulator will have a low runtime error.

3) Researchers must decide what aspects of simulation are important to them: There is in inherent trade-off between simulator performance, accuracy, and flexibility. Having a high-performance and highly configurable simulator is often desirable, because it allows for quick evaluation time and ease of implementation. This flexibility will invariably lead to inaccuracy. Therefore, given their constraints, each engineer must decide how much specificity to be put into their simulator of choice, and they need to be aware of the specification and abstraction errors present in modern simulators.

4) Efforts need to be made to capture the behavior of emerging workloads: Interactive workloads are now being used in simulation, however modern simulators do not model many of the devices they utilize, which leads to large runtime error. The GPU is a prime example of this—we have observed runtime error of up to 3× when running BBench [11] on gem5. This is largely due to the lack of a GPU model in gem5, which causes the system to resort to software rendering. If we do not care about the behavior of individual devices, they may be abstracted away while maintaining accuracy. This intentional abstraction error will allow for simplified and high-performance modelling of these advanced systems, while having low error.

VIII. Conclusions

Because computer architects rely so heavily on architectural simulators it is important that we understand their inherent error and its sources. In this work we have attempted to understand and quantify error in a modern full-system simulator. We do this by validating the gem5 simulator against a real hardware platform: the VExpress development board. Using only publicly available information, we measure the error of a system configured specifically to match our hardware platform. By configuring the system we are able to achieve a MPE for runtime of 5% and a MAPE for runtime of 13%, for the SPEC CPU2006 benchmarks. For PARSEC we are able to achieve a MPE for runtime of -11% and -12% for single and dual-core runs respectively, and a MAPE for runtime of 16% and 17% for single and dual-core runs respectively. We extend our investigation by measuring the accuracy of

several key microarchitectural statistics on gem5; we show that on average, most statistics are accurate to within 20% on average. Finally, we show that when running multi-threaded benchmarks, gem5's scaling is accurate to within 1% on average.

ACKNOWLEDGMENTS

The work presented in this paper was sponsored by Defense Advanced Research Projects Agency (DARPA) under agreement #HR0011-13-2-0006. We thank the reviewers for their feedback.

REFERENCES

[1] Christian Bienia et al. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". In: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. 2008, pp. 72–81.

[2] Nathan Binkert et al. "The gem5 Simulator". In: *SIGARCH Computer Architecture News* 39.2 (2011), pp. 1–7.

[3] N.L. Binkert et al. "The M5 Simulator: Modeling Networked Systems". In: *IEEE Micro* 26.4 (2006), pp. 52–60.

[4] B. Black and J.P. Shen. "Calibration of Microprocessor Performance Models". In: *Computer* 31.5 (1998), pp. 59–65.

[5] Doug Burger and Todd Austin. "The SimpleScalar Tool Set, Version 2.0". In: *SIGARCH Computer Architecture News* 25.3 (1997), pp. 13–25.

[6] A. Butko et al. "Accuracy Evaluation of GEM5 Simulator System". In: *the proceedings of the 7th International Workshop on Reconfigurable Communication-Centric Systems-on-Chip (ReCoSoC)*. 2012, pp. 1–7.

[7] *Cortex-A15 MPCore Technical Reference Manual, Revision: r3p3*. ARM.

[8] Rajagopalan Desikan, Doug Burger, and Stephen W. Keckler. "Measuring Experimental Error in Microprocessor Simulation". In: *the proceedings of the 28th annual International Symposium on Computer Architecture (ISCA)*. 2001, pp. 266–277.

[9] gem5. *gem5 Wiki*. URL: http://www.gem5.org (visited on 12/11/2013).

[10] Jeff Gibson et al. "FLASH vs. (Simulated) FLASH: Closing the Simulation Loop". In: *the proceedings of the ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2000, pp. 49–58.

[11] A. Gutierrez et al. "Full-System Analysis and Characterization of Interactive Smartphone Applications". In: *the proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC)*. 2011, pp. 81–90.

[12] John L. Henning. "SPEC CPU2006 Benchmark Descriptions". In: *SIGARCH Computer Architecture News* 34.4 (2006), pp. 1–17.

[13] Imperas. *OVPsim*. URL: http://ovpworld.org (visited on 12/11/2013).

[14] A. Kivity et al. "kvm: the Linux Virtual Machine Monitor". In: *the proceedings of the 2007 Linux Symposium*. 2007, pp. 225–230.

[15] Travis Lanier. *Exploring the Design of the Cortex-A15 Processor*. URL: http://www.arm.com/files/pdf/at-exploring_the_design_of_the_cortex-a15.pdf (visited on 12/11/2013).

[16] Chih-Chieh Lee, I.-C.K. Chen, and T.N. Mudge. "The Bi-Mode Branch Predictor". In: *the proceedings of the 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1997, pp. 4–13.

[17] P.S. Magnusson et al. "Simics: A Full System Simulation Platform". In: *Computer* 35.2 (2002), pp. 50–58.

[18] Milo M. K. Martin et al. "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset". In: *SIGARCH Computer Architecture News* 33.4 (2005), pp. 92–99.

[19] John D. McCalpin. "Memory Bandwidth and Machine Balance in Current High Performance Computers". In: *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), pp. 19–25.

[20] L. McVoy and C. Staelin. "lmbench: Portable Tools for Performance Analysis". In: *the proceedings of the 1996 USENIX Annual Technical Conference*. 1996, pp. 279–294.

[21] A. Patel et al. "MARSS: A Full System Simulator for Multicore x86 CPUs". In: *the proceedings of the 48th Design Automation Conference (DAC)*. 2011, pp. 1050–1055.

[22] M. Rosenblum et al. "Complete Computer System Simulation: The SimOS Approach". In: *IEEE Parallel Distributed Technology* 3.4 (1995), pp. 34–43.

[23] Ali G. Saidi et al. "Performance Validation of Network-Intensive Workloads on a Full-System Simulator". In: *the proceedings of the first annual Workshop on Interaction between Operating System and Computer Architecture (IOSCA)*. 2005, pp. 33–38.

[24] Timothy Sherwood et al. "Automatically Characterizing Large Scale Program Behavior". In: *the proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2002, pp. 45–57.

[25] Jim Turley. "Cortex-A15 "Eagle" Flies the Coop". In: *Microprocessor Report* (Nov. 2010).

[26] *TwinDie DDR2 SDRAM: Mt47H512M8-32Megx8 x8 Banks x2 Ranks*. Micro.

[27] T.F. Wenisch et al. "SimFlex: Statistical Sampling of Computer System Simulation". In: *IEEE Micro* 26.4 (2006), pp. 18–31.

[28] Roland E. Wunderlich et al. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling". In: *the proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)*. ISCA '03. 2003, pp. 84–97.

[29] M.T. Yourst. "PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator". In: *the proceedings of the 2007 IEEE International Symposium on Performance Analysis of Systems Software (ISPASS)*. 2007, pp. 23–34.