Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations

Joshua J. Yi, *Member*, *IEEE*, and David J. Lilja, *Fellow*, *IEEE*

Abstract—Simulators have become an integral part of the computer architecture research and design process. Since they have the advantages of cost, time, and flexibility, architects use them to guide design space exploration and to quantify the efficacy of an enhancement. However, long simulation times and poor accuracy limit their effectiveness. To reduce the simulation time, architects have proposed several techniques that increase the simulation speed or throughput. To increase the accuracy, architects try to minimize the amount of error in their simulators and have proposed adding statistical rigor to their simulation methodology. Since a wide range of approaches exist and since many of them overlap, this paper describes, classifies, and compares them to aid the computer architect in selecting the most appropriate one.

Index Terms—Simulation, modeling of computer architecture, measurement techniques, modeling techniques, measurement, evaluation, modeling, simulation of multiple-processor systems.

1 Introduction

For almost all computer architecture research and design, quantitative evaluation of future architectures is possible only by using simulators. Simulators reduce the cost and time of a project by allowing the architect to quickly evaluate the performance of a wide range of architectures.

Simulators, however, are limited by two problems, the first of which is slow simulation speed. Since simulators usually simulate programs that are supposed to be run on silicon and since the speed of most cycle-accurate simulators is orders of magnitude slower, simulating all benchmarks in a suite to completion is virtually impossible. To minimize the simulation time, architects typically simulate only a subset of benchmarks, use reduced input sets, or simulate what is ostensibly a representative set of intervals from the program.

The second problem is poor accuracy. This problem is particularly pernicious since virtually every decision that the architect makes can degrade the accuracy and the amount of error is impossible to determine ex post facto. Overall, the accuracy is affected by: the simulator's accuracy, the soundness of the simulation methodology, the representativeness of the benchmarks, and the simulation technique that is used. In this paper, *simulation methodology* is the general term that describes how the architect sets up and runs the simulations. By contrast, a *simulation technique* is the approach the architect uses to perform the simulations, e.g., reduced input sets and sampling.

Given the importance of simulators, this paper surveys prior work that proposed approaches to improve computer architecture simulation methodology and techniques for cycle-accurate simulation. This work can be classified into the following categories: simulator validation, parameter value selection, benchmark selection, simulation techniques, and performance analysis.

2 An Overview of Popular and Emerging Simulators and Benchmarks

To provide a common starting point, this section describes a few popular simulators and benchmark suites. Section 2.1 describes the simulators, while Section 2.2 describes the benchmark suites. The goal of this section is to give a short overview of the simulators and benchmarks suites that are typically used with the simulation methodologies and techniques described in Section 3.

2.1 Simulator

The simulators that are described in this section are classified into single-processor performance simulators (SimpleScalar [6], [57]), full-system simulators (Simics [69], [120]), single-processor power consumption simulators (Wattch [12]), multiprocessor performance simulators (RSIM [46], [95], [26]), and modular simulators (Liberty [111], [112], [113]).

2.1.1 SimpleScalar: A Single-Processor Performance Simulator

The SimpleScalar toolkit was initially released in 1995 [6], while the latest version of SimpleScalar, version 3.0, was released in 1998. Since then, SimpleScalar has become one of the most popular simulators, if not the most popular.

- *J.J. Yi is with Freescale Semiconductor, Inc., 7700 West Parmer Lane, MD: PL30, Austin, TX 78729. E-mail: joshua.yi@freescale.com.*
- *D.J. Lilja is with the Department of Electrical and Computer Engineering, University of Minnesota-Twin Cities, 200 Union Street S.E., Minneapolis, MN 55455. E-mail: lilja@ece.umn.edu.*

Manuscript received 16 Jan. 2004; revised 1 June 2005; accepted 18 Sept. 2005; published online 20 Jan. 2006.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0019-0104.

SimpleScalar consists of simulators and the tools that support the simulators (e.g., a *gcc* compiler, its libraries, an assembler, and a linker). The different simulators have a wide range of speed versus detail trade-offs.

Due to its level of detail, `sim-outorder` is usually the simulator of choice for architects. This simulator models a superscalar processor with a five-stage pipeline. User-configurable parameters include: the type/size of the branch predictor, the number of load-store queue entries, and the cache configuration. One unique structure in `sim-outorder` is the register-update unit, which is a combination of the reorder buffer (ROB) and the functional units' reservation stations.

Version 4.0 of `sim-outorder`—called MASE (Micro Architectural Simulation Environment) [57]—is currently under development. Its notable features include: 1) splitting the RUU into a ROB and separate reservation stations, 2) a "checker" unit that verifies the correct execution of the program to support speculation, and 3) a more realistic memory interface.

Downloads and more information about SimpleScalar can be found at http://www.simplescalar.com.

2.1.2 Simics: Full-System Simulation

Although processor simulators like SimpleScalar model the microarchitecture, they do not model the entire system or run an operating system (OS). Cain et al. [14] show that omitting the OS can produce errors as high as 100 percent in the simulation results for the SPEC CPU 2000 benchmarks. To address these problems, architects use full-system simulators that model the entire system in sufficient depth to run the OS.

Simics is a full-system simulator that is available from Virtutech AB [69], [120]. Simics supports a wide range of instruction sets, including Alpha, PowerPC, UltraSparc, and x86-64, and includes device models, such as graphics cards, ethernet cards, and bus controllers, that are detailed enough to run the actual device drivers. Also, Simics can boot and run unmodified operating systems such as Linux, Solaris, and Windows XP. Simics' more interesting features include breakpoint and replay capabilities, support for program checkpoints, and using Simics Central to simulate multiple processors on different physical machines.

Simics includes three processor models that have different levels of simulation detail and speed [120]. The fastest simulator models an in-order processor with single-cycle execution latencies. The slowest simulator is a detailed out-of-order processor. With this processor model, the user specifies a detailed timing model and how instructions should be executed (e.g., how many instructions can be executed out-of-order and speculatively executed). The third simulator is a scaled back version of the fully specified processor that bypasses the user-visible API and reduces the number of instruction execution options [120] to improve the simulation speed.

Downloads and more information about Simics can be found at http://www.virtutech.com.

To reduce the development time of full-system simulators, Mauer et al. [73] proposed *timing-first simulation*. Compared to conventional simulators, timing-first simulators have several advantages. First, they have a lower development time since they do not fully model all aspects of the system. Although the system is not fully modeled, their timing-first simulator still executed 99.99 percent of all instructions correctly while incurring a maximum error of 4.8 percent only. Second, since the functional part of simulator constantly checks for correct program execution, errors that would normally cause a conventional simulator to crash are detected and repaired. Third, by focusing on the timing part of the simulator first, speculative execution is modeled more precisely.

Other well-known full-system simulations include: SimOS from Stanford [93], [94], [126], Mambo from IBM [98], [102], and SimOS-Alpha from DEC [101].

2.1.3 Wattch: A Single-Processor Power Consumption Simulator

One of the limiting factors for future processors is their power dissipation. To estimate a processor's power consumption, architects use simulators such as Wattch [12]. Wattch was developed at Princeton and is based on `sim-outorder`, version 3.0. To calculate the power consumption for a component, Wattch uses the formula: $P = C * V_{dd}^2 * a * f$, where C is the capacitance along all paths within the component, V_{dd} is the supply voltage, a is the component's internal switching activity, and f is the clock frequency. The energy consumed each cycle is simply the sum of the energy that was consumed by each component. Wattch models V_{dd} and f as constants since they depend on the process technology. Wattch uses template formulas to calculate the capacitance for four types of components: array structures (e.g., caches, etc.), fully associative content addressable memory structures (e.g., ROB, LSQ, etc.), combinational logic (e.g., functional units, etc.), and the clock distribution network. The amount of switching activity is determined by monitoring the actual bit transitions within the component during simulation. The results in [12] show that the distribution of energy consumption across all major processor components is very close to that of the Pentium Pro and the Alpha 21264, two processor architectures that are similar to the base Wattch architecture.

Wattch can be downloaded from http://www.eecs.harvard.edu/~dbrooks/wattch-form.html.

Although Wattch is the most widely used power consumption simulator [15], several other simulators estimate the processor's power consumption, including PowerTimer [13] from IBM, SimWattch [15] from the USC, PowerAnalyzer [90] from Michigan, PowerImpact from UCLA [91], and SimplePower [119] from Penn State. Of these simulators, SimWattch is particularly interesting since it integrates a full-system simulator (Simics) with Wattch. This is important since omitting operating system effects leads to overestimation of the performance results while underestimating the power consumption results [15].

2.1.4 RSIM: A Multiprocessor Performance Simulator

RSIM—the Rice Simulator for ILP Multiprocessors—was developed at Rice and released to the public in 1997 [46], [95], [26]. Key RSIM features include out-of-order issue, register renaming, branch prediction, nonblocking caches, multiple cache-coherence protocols, and user configurable

parameters such as instruction window size, cache sizes and latencies, and flit size and delay [46]. Modeling ILP-level (i.e., intraprocessor) features is important even when simulating multiprocessor systems since these features may interact with thread-level (i.e., interprocessor) features. To illustrate this problem, Hughes et al. [46] evaluated the performance of *read miss clustering* using RSIM and with a multiprocessor simulator that did not model the ILP-level features. Their simulation results show that disregarding the ILP-level features overestimates the execution time by as much as 132 percent. Even when execution times were similar, ignoring the ILP-level features still overestimated the number of cycles due to memory access. Finally, and perhaps worst of all, the results show that not modeling ILP-level features could distort the simulation results to such a degree that the architect would conclude that read miss clustering enhancement yields very little performance improvement when it, in actuality, yields significant performance benefits.

Downloads and more information about RSIM can be found at: http://rsim.cs.uiuc.edu.

Other multiprocessor simulators include: Talisman [8], Tango [23], a multiprocessor version of SimpleScalar [70], Wisconsin Wind Tunnel II [80], Augmint [82], MINT [118], GEMS [37], [71], and M5 [68].

2.1.5 Modular Simulators

Although processors and systems are highly parallel, the simulators that model their behavior are typically written with sequential programming languages. The consequences of this parallel-to-sequential mapping are that the implementation and debugging of the simulator can be very time-consuming, its accuracy may suffer due to insufficient detail, and component reuse is unlikely.

To minimize the effects of these problems, architects have proposed using modular simulators such as the Liberty Simulation Environment (LSE) [111], [85], [112], [113]. Developed at Princeton, Liberty maps each hardware component to a single software function; by instantiating those components and specifying its connections, architects can hierarchically build more complex processor components. Hierarchically building processor components has at least two key advantages. First, by building and testing lower-level components hierarchically, the architect can easily build large libraries of accurate components. Second, the architect can quickly and easily build a wide range of models, which allows for efficient design space exploration and/or the examination of more unusual architectures. To ensure that future components will seamlessly interface with current components, the LSE stipulates the control interface between LSE components; in particular, `enable` and `ack` handshaking signals are used to synchronize the data transfer between components.

More information and downloads about the LSE can be found at http://liberty.princeton.edu.

In addition to the LSE, other recent modular simulators include: Asim [35], EXPRESSION [76], LISA [84], and Microlib [87].

2.2 Benchmark Programs

This section divides benchmark suites into three categories: general purpose computing (SPEC CPU 2000 [45]), embedded systems benchmarks (MiBench [42]), and other benchmark suites.

2.2.1 SPEC CPU 2000: A General Purpose Benchmark Suite

Benchmark suites released by the Standard Performance Evaluation Corporation (SPEC) are very popular with architects, to the point where the SPEC CPU 2000 benchmark suite [45] has become the de facto benchmark suite for general-purpose computing. SPEC released CPU-intensive benchmark suites in 1989, 1992, 1995, and 2000 [105].

To build a benchmark suite, SPEC solicits candidate benchmarks from SPEC member companies and the public and chooses them using three criteria [105]. First, the benchmark has to be portable across all SPEC hardware architectures and to various operating systems. Since these are CPU-intensive benchmarks, the second criterion is that the benchmarks should not include a measurable amount of I/O, networking, or graphics. Additionally, to ensure that the benchmark does not focus on disk activity, the memory footprint of the benchmark should fit into 256MB of memory without swapping. Third, less than 5 percent of the benchmark's runtime should be spent executing non-SPEC code. In SPEC CPU 2000, there are 12 integer and 14 floating-point benchmarks. The applications for the integer benchmarks range from compression to word processing, while the floating-point applications range from computational chemistry to nuclear physics accelerator design.

More information about all SPEC benchmark suites can be found at http://www.spec.org.

Ideally, the benchmarks in a suite completely cover the application space with as little redundancy as possible and where each benchmark is not "fragile" (its performance does not dramatically improve by a trivial optimization). Vandierendonck and De Bosschere [116] used principal component analysis (PCA) to determine if the SPEC CPU 95 and 2000 benchmarks were fragile and/or "eccentric." Benchmarks are eccentric if their characteristics significantly differ from other benchmarks and, for this reason, when constructing a benchmark suite, these benchmarks should be added to the suite to cover potential "gaps" in the space of benchmark characteristics. Their results showed that both suites have fragile and eccentric benchmarks, although SPEC CPU 2000 has more eccentric benchmarks.

Phansalkar et al. [88] used PCA to characterize and compare four successive generations of SPEC CPU benchmarks, i.e., SPEC CPU 89, 92, 95, and 2000. Their results showed that, other than dramatic increases in the dynamic instruction count and increasingly poor temporal data locality, fundamental program characteristics such as branch and ILP characteristics are generally static.

2.2.2 MiBench: Embedded Benchmark Suite

Although SPEC CPU 2000 is the most popular benchmark suite, its benchmarks are not very representative of the embedded system applications. Therefore, to provide realistic embedded system benchmarks, architects construct

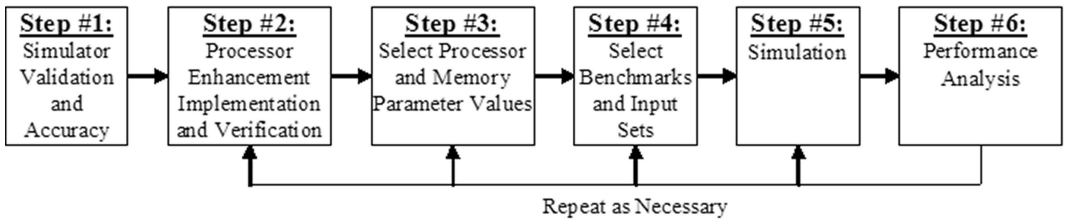

Fig. 1. The simulation process in computer architecture research and design.

benchmark suites that explicitly represent embedded system applications; one such example is MiBench [42] from the University of Michigan.

In MiBench, there are six types of benchmarks:

1. automotive and industrial control (six benchmarks),
2. consumer devices (eight),
3. office automation (five),
4. network (two),
5. security (seven), and
6. telecommunications (seven).

Benchmarks in the first category represent applications that are used for air bag controllers and engine performance monitors, while benchmarks in the second category represent the programs that are present in scanners, digital cameras, and PDAs. Benchmarks in the third category represent the applications that are found in printers, copiers, and fax machines. Benchmarks in the fourth and fifth categories represent programs for shortest path calculations and tree and table lookups and for encryption, decryption, and hashing, respectively. Finally, applications that consist of voice encoding/decoding, along with benchmarks for frequency analysis and checksum calculation, are in the sixth category.

MiBench can be downloaded from http://www.eecs.umich.edu/mibench.

Other embedded benchmark suites include: EEMBC (EDN Embedded Microprocessor Benchmark Consortium) [124], [33] for automotive, consumer and digital entertainment (i.e., multimedia), java, networking, office automation, and telecommunications applications; Commbench [127], NetBench [74], [75], and NpBench [61] for network applications; and MediaBench [62] for multimedia applications.

2.2.3 Miscellaneous Benchmark Suites

In addition to the above benchmark suites, other suites exist to measure the performance of other processor architectures. Scientific benchmark suites include SPLASH (Stanford Parallel Applications for Shared Memory) and SPLASH2 [103], [128], SPEC HPC 2002 and SPEC OMP [106], ASCI [5], NAS [81], and SciMark [97]. Java benchmarks include: SPEC JVM98 [108] and SPEC jbb2000 [107] from SPEC, Java Grande [72], [48], and VolanoMark [121]. Benchmarks from the Transaction Processing Performance Council [110] measure the rate at which a system can process business-related transactions and access database system programs. Other miscellaneous benchmark suites include 3DMark [1] for 3D applications, BioBench [4] for bioinformatics benchmarks, SYSmark [109] for office productivity software, and lmbench [65] and HINT [41] measure specific processor components.

For more in-depth information about these benchmarks and others, the reader is referred to [123], [49].

3 Simulation Methodologies and Techniques

For architects, the ideal simulator faithfully models all aspects of the processor, is very easy to modify, and has a very fast simulation speed [6]. However, finding the proper level of abstraction [11] is important since increasing the simulator's accuracy almost always comes directly at the expense of speed. In addition, the accuracy of the results also depends on the simulation methodology, i.e., how the results were generated. The remainder of this section describes previous work that proposed approaches to improve simulation methodology and accuracy, or new simulation techniques.

3.1 The Simulation Process

The *simulation process* is the sequence of steps that the architect must perform to run and analyze the simulations. In this paper, we divide the simulation process into the six steps shown in Fig. 1. Of these six steps, no previous work focused specifically on improving the accuracy of implementing a processor enhancement (Step 2). Consequently, the following subsections focus on Steps 1 and 3-6 only.

3.2 Step #1: Simulator Validation and Accuracy

The starting point for producing accurate results is to use a simulator that faithfully models the hardware. Black and Shen [9] iteratively improved the accuracy of their performance model by comparing the simulated cycle count against the cycle count of the actual hardware. Even after a long period of debugging, their results show that modeling, specification, and abstraction errors were still present in their simulation model. Their work showed the need for extensive, iterative validation before the results from a performance model can be trusted.

Gibson et al. described the errors in the FLASH simulator when compared to the custom-built FLASH multiprocessor system [38]. Their results showed that 1) most simulators can accurately predict the trend if all of the important components have been accurately modeled, 2) a faster simulator that uses a scaling factor for the results often predicts the processor's performance more accurately than a more complex simulator, and 3) the error of some simulators exceeded 30 percent, which is higher than the speedups that are often reported for processor enhancements.

Desikan et al. [24], [25] measured the amount of error, as compared to the Alpha 21264 processor, that was present in

the Alpha version of the SimpleScalar simulator. Their results showed that unvalidated simulators generally report higher IPCs than the hardware that they model. This is not surprising since unvalidated simulators will typically tend to underestimate the complexity of implementing some microarchitectural features that have a large impact on the IPC.

Cain et al. [14] measured the effect of the OS and the effects of I/O on simulator accuracy. Their results showed that omitting an OS introduced errors as high as 100 percent for the SPECint 2000 benchmarks. Although non-SPEC code only accounts for 5 percent of the instructions, the high amount of error is probably due to the long time needed to handle DMA and I/O and secondary effects such as the cache and branch predictor "pollution" due to executing OS code. Overall, their results showed the need for full-system simulation for increased accuracy and precision.

Collectively, [9], [14], [24], [25], [38], [79] show that: 1) the performance results using unvalidated simulators can be quite different from the target processor and, consequently, cannot be fully trusted and 2) architects need to use detailed simulators to eliminate abstraction errors. These conclusions are especially significant for architects since the performance of an enhancement is usually based only on the simulation results. Consequently, it is important that future simulators be very detailed and undergo a significant amount of validation.

3.3 Step #3: Select Processor and Memory Parameter Values

In the third step of the simulation process, the architect chooses values for the processor and memory parameters. Obviously, choosing an inappropriate set of parameter values can significantly affect the simulation results. However, architects cannot choose each parameter independently of the other parameters since many of the parameters interact in unexpected ways and since these interactions can significantly impact the results. Unfortunately, determining which parameters are significant is very time consuming due to the large number of parameters in most simulators.

To address this problem, Yi et al. [132] proposed using the statistical Plackett and Burman (P&B) design to determine the most significant parameters. A P&B design is superior to the Analysis of Variance (ANOVA) design [63] in that it quantifies the effect of single parameters and selected interactions only, in $O(\mathrm{N})$ simulations, where N is the number of parameters. By contrast, the ANOVA design requires $O(2^{\mathrm{N}})$ simulations to determine the effect of all parameters and interactions. Since the P&B design can identify the most significant parameters, the difficulty of choosing parameter values is greatly reduced. In particular, Yi et al. [132] recommend using the following steps to choose the values for processor and memory parameters:

1. Determine the most critical parameters using a Plackett and Burman design.
2. Iteratively perform sensitivity analyses for each critical parameter using ANOVA.
3. Choose final values for the critical parameters based on the results of the sensitivity analyses.
4. Choose final values for the noncritical parameters based on an appropriate source, such as using the values that are common in commercial processors, for instance.

3.4 Step #4: Select Benchmarks and Input Sets

For most research and design projects, simulating all of the benchmarks in a suite is not possible due to finite computing resources. To reduce the simulation time, architects typically simulate only a subset of the suite's benchmarks. However, choosing a nonrepresentative subset can skew the results. As a result, instead of selecting benchmarks at random or based on fuzzy, unproven characteristics, architects should select benchmarks in a rigorous, quantitatively justifiable manner.

Citron [18] analyzed how and why architects selected benchmarks from the SPEC CPU 2000 suite. His results showed that most papers publish results for only a few selected benchmarks. For example, in the ISCA 2002 conference, of the 23 papers for which SPEC CPU 2000 was an appropriate choice, only four and six papers used all 12 integer and 14 floating-point benchmarks, respectively. Even then, when a subset was used, for the most prestigious conferences (ISCA 2001-2002, MICRO 2001-2002, and HPCA 2002-2003), less than a third of the papers that used a subset of SPEC CPU 2000 explained why they selected that subset.

Yi et al. [132] classified benchmarks based on the similarity of their performance bottlenecks, which they determined using a P&B design. The parameters were ranked in descending order of importance, where a rank of "1" was assigned to the most significant parameter, and then vectorized. Two benchmarks are similar if the Euclidean distance between their vectors of ranks is small. By iteratively computing the Euclidean distance between each pair of vectors (benchmarks) and, subsequently, groups of vectors, all benchmarks were clustered.

To characterize a set of benchmarks, Eeckhout et al. [27], [28] first gathered a set of metrics such as the instruction mix, branch prediction accuracy, cache miss rates, and basic block lengths for each benchmark and input set pair. After gathering these metrics, they used PCA to determine the principal components for each pair and then clustered the pairs based on their principal components. Phansalkar et al. [88] also used PCA to characterize the benchmark and input set pairs, but they used k-means clustering and the Bayesian Information Criterion instead to cluster the benchmark and input set pairs. Using this approach, they were able to represent the entire SPEC CPU 2000 benchmark suite with eight benchmarks, with less than 5 percent error for 8 and 16-way processors.

Although these papers differ in their benchmark classification approaches, their basic motivation is the same, namely, they propose statistically rigorous approaches of benchmark classification. After classifying the benchmarks into N different groups, where N is the maximum number of benchmarks that can be simulated, the architect needs only to select one benchmark from each group to form a subset of benchmarks to represent the entire suite. The key difference between [132] and [27], [28], [88] is that [132] classifies benchmarks based on their performance bottlenecks while

[27], [28], [88] classify them based on architectural performance metrics.

Finally, Vandierendonck and De Bosschere [117] evaluated the accuracy of four clustering algorithms for a varying number of clusters when subsetting the SPEC CPU 2000 suite. Specifically, they examined k-means clustering, linkage clustering, and two PCA-based methods. Their results show that none of these algorithms consistently produces a representative subset of benchmarks. The most reliable technique was the backwards algorithm based on PCA.

3.5 Step #5: Simulation

After selecting the parameter values and benchmarks, the architect runs the simulations. Unfortunately, for benchmark suites such as SPEC CPU 2000, the simulation time is measured in weeks. To combat this problem, architects have proposed several different techniques with a wide range of accuracies and speeds.

3.5.1 Reduced Input Set Simulation Techniques

One obvious approach to reduce the simulation time is to modify the input set such that the benchmark executes fewer instructions, while, ideally, having the same characteristics as the full input set. Reduced input sets have two advantages over other simulation techniques. First, the benchmark runs to completion, which means that the simulator executes all pieces (e.g., initialization) of the program. Second, using reduced input sets typically does not require any changes to the simulator.

KleinOsowski and Lilja [53] created the MinneSPEC reduced input set for the SPEC CPU 2000 benchmarks. They were created by modifying command-line parameters, truncating the `reference` input set, or creating a completely new input set. For each benchmark, they tried to create three reduced input sets, `small`, `medium`, and `large`, that executed approximately 100M, 500M, and 1,000M, respectively, dynamic instructions. Their results showed that 18 of the 32 benchmark and input set pairs had execution profiles that were statistically similar (using the chi-squared test) to the `reference` input set. Furthermore, 25 pairs had statistically similar instruction mixes. However, their results showed that the reduced input set cache miss rates were very different for some benchmarks. Not surprisingly, this difference translates into poor IPC accuracy [133]. Eeckhout et al. [29] show that the large input set is typically the most `reference`-like.

Alameldeen et al. [2] created reduced input sets for TPC-C based on the metric of transaction throughput. Since the initial throughput was too low, they proceeded to:

1. tune several kernel and database parameters,
2. relieve the I/O bottleneck by partitioning the database across five disks,
3. increase the number of warehouses without increasing the database size to decrease the amount of contention to the same warehouse table, and
4. add more users.

These changes resulted in a 12-fold increase in the transaction throughput, which made the workload more representative of OLTP workloads.

The key conclusion from these two papers is that creating a reduced input set that accurately replicates the characteristics of the full input set is a very time-consuming and user-intensive task, especially so since reduced input sets need to be recreated for each new suite.

3.5.2 Truncated Execution Simulation Techniques

Truncated execution is another technique that can significantly reduce the simulation time. With this technique, instead of simulating the entire benchmark, the architect simulates `Z` million contiguous instructions only from somewhere in the benchmark. The three main variations of truncated execution are Run `Z`, Fast-Forward `X` + Run `Z` (FF `X`+ Run `Z`), and Fast-Forward `X` + Warmup `Y` + Run `Z` (FF `X` + WU `Y` + Run `Z`).

With Run `Z`, the architect simulates only the first `Z` million instructions of the benchmark and then stops the simulation. The simulation time is proportional to the value of `Z`. One problem with this approach is that the entire simulation could be spent executing initialization code, which is not representative of the entire benchmark. Increasing the value of `Z` to avoid executing the initialization code only erodes the benefit of this technique by increasing the simulation time.

To potentially simulate a more representative slice of the benchmark, the FF `X` + Run `Z` variation fast-forwards (i.e., performs functional simulation only) through the first `X` million instructions and then performs detailed simulation of the next `Z` million only. To minimize the effect of a "cold" processor and memory subsystem (MSS), which is the state after fast-forwarding, the architect can "warmup" the processor and MSS by performing detailed simulation for `Y` million instructions after fast-forwarding. Then, for the simulation results, the architect can track the statistics over the next `Z` million instructions of detailed simulation only. Depending on the value of `X`, the simulation time of fast-forwarding can be significantly longer than the Run `Z` time.

3.5.3 Processor Warmup Approaches

Although warmup improves the simulation accuracy, it also increases the simulation time, accounting for 50 percent or more of the time [43]. Therefore, reducing the warmup time can significantly decrease the simulation time.

Haskins and Skadron [43] proposed Minimal Subset Evaluation (MSE) to minimize the number of warmup instructions. MSE probabilistically determines a minimal set of cache accesses that need to occur to produce a sufficiently accurate cache state. Their results showed that, for a 99.9 percent confidence level, MSE decreased the warmup time by an average of 47 percent with only a 0.3 percent IPC error. At 95 percent, the warmup time decreased by an average of 60 percent with a 0.4 percent IPC error.

Their following approach—Memory Reference Reuse Latency (MRRL) [44]—used the time between memory references to determine the number of warmup instructions. More specifically, this approach determines the number of warmup instructions such that N percent of them have reuse latencies (number of instructions between accesses to the same address) less than the number of warmup instructions. This approach exploits the fact that memory references exhibit temporal locality since addresses

that are not accessed in the warmup period will probably not be accessed again during detailed simulation. Their results showed that this approach reduces the warmup time by an average of 90.6 percent with less than 1 percent IPC error.

One potential weakness of MRRL is that it computes most of the reuse latencies from the instructions in the warmup period, instead of basing them on the instructions in the detailed simulation period. Eeckhout et al. improve on this approach by calculating the reuse latency for each memory reference in the detailed simulation period back into the warmup period [30]. Van Biesbrouck et al. describe a faster version of MRRL in [115].

One key feature of these approaches is that they can be added to any fast-forwarding or sampling-based technique. Their drawback, however, is that they determine the warmup period offline. Luo et al. [67] dynamically determine the warmup time by using two criteria: 1) after all cache lines have been accessed at least once or 2) after a certain, user-defined, number of cache accesses. The former approach is more appropriate for smaller caches and/or programs with smaller memory footprints, while the opposite is true for the latter.

Other warmup approaches include: 1) assuming that each cache entry hits when accessed for the first time [129], 2) reusing the cache state as it existed at the end of the last interval [51], and 3) continually updating the caches and branch predictor during nondetailed simulation [130].

3.5.4 Sampling Simulation Techniques

Population sampling is a statistical technique that is used to infer the characteristics of a larger set by extrapolating from the characteristics observed in a subset. The key to good results is to ensure that the sample accurately reflects the overall population. Three primary sampling techniques, representative, periodic, and random, have been proposed for use in simulation-based computer architecture research studies.

Representative Sampling Simulation Techniques. Representative sampling attempts to extract a subset of a benchmark's instructions that matches its overall execution behavior. In computer architecture, representative sampling can be applied by determining a group of intervals that could be substituted for the entire benchmark.

To choose a representative instruction interval of 50M instructions, Skadron et al. [104] quantified the branch misprediction rate for 1M instruction intervals. Then, by examining branch misprediction rates over the entire benchmark's execution, they selected a representative 50M interval. Their results showed that the accuracy heavily depends on the warmup length.

Instead of using the branch misprediction rate—which is configuration dependent—Lafage and Seznec [55] characterized each 1M interval based on the amount of temporal and spatial locality intrinsic to that interval. After clustering the intervals together based on the similarity of their temporal and spatial locality, they selected the interval that was closest to the centroid of the cluster as the cluster's representative. Their results show that the relative error in the cache miss rate averaged 1.52 percent while requiring only 3.34 percent of the simulation time for SPEC CPU 95.

For SimPoint, Sherwood et al. [100], [99] characterize each instruction interval, or simulation point, using basic block vectors (BBV). For each interval, the BBV contains the execution frequency of each basic block multiplied by the number of instructions in that block. (Instead of using BBVs, Lau et al. [58] used other metrics, such as loop branches, register usage, etc., to characterize each interval.) After using linear random projection to reduce the dimensionality of each BBV, they used k-means clustering to cluster the intervals based on their Euclidean distance and then selected the interval closest to the centroid of each cluster. To use SimPoint, the architect simulates each selected interval and multiplies the results for that interval by its corresponding weight, before summing the weighted products together. Their results showed that using multiple intervals dramatically decreased the simulation time with an average IPC error of 3 percent. Perelman et al. [86] improved the basic SimPoint algorithm in two ways. First, their algorithm estimates the amount of CPI error for a given confidence interval and then selects the set of intervals with the least amount of CPI error. Second, to reduce the fast-forwarding time, their algorithm chooses intervals that occur earlier in the program. Finally, Lau et al. [59] extended SimPoint to allow for variable-length intervals (to better align the intervals to the phase boundaries), while Van Biesbrouck et al. [114] extended SimPoint to accurately guide simultaneous multithreading simulations.

For EXPERT, Liu and Huang [64] partitioned the benchmark into sections of static code, characterized each section with several different metrics (CPI, basic block size, branch prediction accuracy, etc.) with multiple input sets and processor configurations, and, then, based on the characterization results, chose which instances of static code to simulate. Their results showed that this approach dramatically reduces the simulation time (excluding the one-time characterization runs) while incurring an average error of 1 percent.

Periodic Sampling Simulation Techniques. Periodic sampling simulates selected portions of the dynamic instruction stream at fixed intervals. The sampling frequency and the interval length determine the simulation time.

SMARTS (Sampling Microarchitectural Simulation) [130] is a recent example of periodic sampling. To minimize the explicit detailed warmup period, SMARTS uses *functional warming* (continuous updates of the caches and branch predictor during functional simulation) to keep key components warm. SMARTS estimates the CPI error of the sampled simulation and, if the estimated error is higher than ± 3 percent, for a given confidence level, SMARTS suggests using a higher sampling frequency. Their results showed that SMARTS yields speedups of 36X to 60X while incurring an average error of 0.64 percent in the CPI and 0.59 percent in the energy per instruction. TurboSMARTS improves the simulation speed of SMARTS by using checkpoints [125]. They minimize the size of the checkpoints by identifying the sequence of instructions that commit during warmup and detailed simulation and store only the state that these instructions access, in addition to storing only correct-path memory accesses.

Random Sampling Simulation Techniques. Random sampling combines the simulation results from N randomly chosen and distributed intervals to produce the overall simulation results. For random sampling, the sampling error falls into two groups: nonsampling and sampling bias. Nonsampling bias occurs when the sampled population differs from the actual population, which, in computer architecture simulations, is primarily due to the cold-start effect [19]; increasing the warmup time or using a more accurate processor state reduces this effect. Sampling bias occurs when the mean of the sampled distribution differs from the population mean, which, in computer architecture simulation, is the result of simulating an insufficient number of instructions. To minimize the effects of sampling bias, architects merely need to increase the number of sampling units or increase the number of instructions in each sampling unit.

A Comparison of Periodic and Random Sampling. If a benchmark has periodic behavior and if the sampling frequency is close to the benchmark's natural frequency, then periodic sampling will have a lower accuracy than random sampling, potentially even when random sampling takes fewer measurements. For example, assume that the IPC graph of a benchmark resembles a sine wave and that periodic sampling happens to take samples at the peak. In this case, periodic sampling will estimate that the benchmark's IPC is equal to the peak IPC. On the other hand, random sampling will yield a more accurate estimate of the benchmark's IPC if enough measurements have been made. While it may be rare that the sampling frequency is similar to the benchmark's frequency, it is more likely that the sampling frequency may be the same as the natural frequency of some of the benchmark's phases. For those phases, periodic sampling will yield poor estimates of the IPC.

However, when the benchmark and sampling frequencies are dissimilar, the accuracy of random sampling will be lower than that of periodic sampling, for the same number of sampling units, if and only if the intrasample variance is greater than the variance of the population as a whole. Since random sampling randomly simulates intervals of instructions throughout the entire duration of the program, some phases of the program could be oversampled at the expense of undersampling other phases. This overweighs the oversampled phases, which decreases the accuracy. In this case, the architect needs to take more samples—which increases the simulation time—to minimize the error. On the other hand, if the sampling frequency is not high enough, then periodic sampling is guaranteed to undersample some/all phases.

With random sampling, each instruction has the same probability of being included within a sample. Consequently, it is reasonable to assume that the distribution of errors for random sampling follows a Gaussian distribution. Similarly, with periodic sampling, each instruction also has an equal probability of being included within a sample if the starting point of the first sample is randomly chosen and there is no synchronization between the sampling function and any repetition of events in the process being sampled. Consequently, it is again reasonable to assume a Gaussian error distribution for periodic sampling. This assumption is further strengthened by ensuring that a large number of samples are taken. In this event, the central limit theorem assures us that the error distribution will, in fact, be Gaussian [63] for both forms of sampling. However, if the number of sampling units is small, then the Gaussian error distribution assumption may be suspect for both random and periodic sampling.

A more important assumption than the Gaussian distribution of errors is the estimate of the variance between sampling units. Periodic sampling will likely overestimate the variance as compared to random sampling, which provides conservative estimates of the precision of the measurements obtained with periodic sampling. This overestimation is good since it forces the experimenter to select more sampling units than are actually necessary to obtain the desired precision, at the cost of longer simulation time.

In summary, both random and periodic sampling have their strengths and weaknesses. However, if a large number of samples is taken using either approach and it can be ensured that the sampling frequency of periodic sampling is asynchronous with respect to program phases, either approach can be used effectively.

Other Sampling Techniques and Papers. Wunderlich et al. [131] analyzed the potential speed and accuracy of stratified sampling. Laha et al. [56] and Conte et al. [20] proposed sampling-based techniques to quickly evaluate cache performance. Iyengar et al. [47] described a new metric, the R-metric, which is based on microarchitectural metrics, that evaluates the representativeness of an instruction trace, while Khalid [52] does the same for the K-metric, which is based on the instruction mix. Ekman and Stenstrom [34] proposed using matched-pair comparison to reduce the simulation time of sampling-based approaches. For benchmarks such as Acrobat Reader, Netscape, and Word, Crowley and Baer [21], [22] show that trace sampling can accurately estimate cache miss ratios provided that the processor is properly warmed up before the start of each sample. Finally, the results in [66] show that, when using sampling, to produce accurate estimates of the speedup, it is not necessary to estimate the CPI to the same level of accuracy (as is needed for accurate CPI estimates).

3.5.5 *Accelerating Simulation: Direct-Execution, Checkpointing, and Parallel-Simulation, Efficient Simulator Implementation Techniques.*

As the dynamic instruction count increases, the speed of functional simulation, i.e., fast-forwarding, limits the potential speedup of sampling-based techniques. To increase the speed of functional simulation, architects can run the benchmark directly on the host machine, instead of simulating the benchmark during the functional simulation period. This approach is called *direct-execution*. Various implementations of direct-execution include [126], [54], [16].

To eliminate the fast-forwarding time, architects can use *checkpoints* to minimize the simulation time. To create a checkpoint, the architect executes the program until the checkpoint and then saves the program state to a checkpoint file. Then, before simulation, the user-visible registers are updated with the contents of the checkpoint file. Not only does checkpointing allow the architect to skip

over fast-forwarding in future simulations, it also permits multiple intervals to be simulated in parallel. Schnarr and Larus [96] showed that memoization (i.e., checkpointing) improved the performance of their cycle-accurate simulator by a factor of 4.9 to 11.9. Barr et al. [7] described how to add memory timestamp records to multiprocessor checkpoints to minimize the warmup time of multiprocessor simulations. Ringenberg et al. [92] propose adding explicit checkpointing support into the benchmark program so that the start of the benchmark binary contains the checkpoint(s). Van Biesbrouck et al. describe a technique to reduce the checkpoint size in [115].

Another approach to improve simulation speed is to parallelize the simulation by distributing the simulation across several processors. However, the downside of this approach is that the simulated processor needs to be warmed up for each distributed simulation interval, which decreases the speedup of distributing the simulation. To minimize the effect of distributed warmup, Girbal et al. [39] dynamically adjust the warmup time by comparing the CPIs of the instructions that overlap two intervals (since the same instructions that are used for warmup on one processor are simulated in detailed on another). When the CPIs are within a user-defined range, the warmup period is over. One key advantage of this approach is that it essentially distributes the simulation workload based on the accuracy. Other approaches to parallel simulation include [17], [31], [36], [60], [83], [89], [122].

Finally, Moudgill [77], [78] describes several implementation techniques to improve the speed of the simulator such as simulating the pipeline in reverse order, building less-detailed processor models at compile time, and techniques to reduce the search time of tag arrays for cache simulation.

For more in-depth information on these simulation techniques and others, see [32].

3.5.6 Comparison of Simulation Techniques

Yi et al. [133] characterized and compared three truncated execution variations, reduced input sets, SimPoint, and SMARTS. They characterized these techniques by their performance bottlenecks (via the P&B design), basic-block execution frequencies, and architectural-level performance metrics (IPC, cache miss rate, and branch prediction accuracy). They also compared the speed versus accuracy and the configuration independence of each technique. Their results showed that, for all three characterizations, truncated execution and reduced input sets had very poor accuracy, as compared to the `reference` input set. Overall, both SimPoint and SMARTS are very accurate; the most accurate SimPoint permutation is multiple 10M simulation points, while the most accurate SMARTS configuration is for a simulation interval of 10,000 instructions. Although SMARTS is more accurate than SimPoint, SimPoint requires less simulation time than does SMARTS, even after accounting for the time needed for profiling, simulation point generation, and checkpoint generation. Finally, of these techniques, SMARTS is the technique that is the most configuration independent since it has the lowest CPI error across a wide range of processor configurations.

Gómez et al. [40] show that the L2 behavior and branch prediction accuracy of several MinneSPEC reduced input sets are not `reference`-like and that the accuracy of truncated execution depends heavily on fast-forwarding to a representative interval.

3.5.7 Minimizing Variability in Multiprocessor Simulation Results

Since most simulators are deterministic, one characteristic of real systems that architects typically do not account for is experimental variability, which may result in incorrect conclusions.

Alameldeen and Wood [3] investigated the potential impact that variability can have on the simulation results. To demonstrate the potential effect of variability, they added a random amount of time to each L2 cache miss and measured its effect on the execution time for increasing L2 cache associativity and ROB entries. Each test case was run 20 times. Their results showed that, due to this variability, an architect could arrive at the wrong conclusion that 31 percent and 26 percent of the time for increasing L2 cache associativity and ROB entries, respectively. To minimize this potential problem, they recommended adding pseudorandom perturbation to the simulations, simulating each test case multiple times, and using confidence intervals and hypothesis testing.

3.6 Step #6: Performance Analysis

The last step in the simulation process is to analyze the effect that an enhancement has. For most studies, this analysis extends only to calculating the speedup or measuring the change in a key metric. Unfortunately, these approaches only give the architect a high-level, net-effect picture.

For a more detailed analysis, Yi et al. [132] proposed using the P&B design to quantify the effect that an enhancement has on the processor's performance bottlenecks. The results of a P&B design can be used to rank the performance bottlenecks with and without the enhancement. By comparing the change in a bottleneck's rank with and without the enhancement, the architect can easily determine if the enhancement relieves or exacerbates each bottleneck.

3.7 Comparing Absolute Accuracy versus Relative Accuracy

Although the absolute accuracy of a simulator/technique is very important, perhaps what is more important, is its *relative accuracy*, i.e., how well each tracks changes in the target value. Therefore, while a simulator/technique may have poor absolute accuracy since it consistently overestimates the IPC, it may have excellent relative accuracy since its estimated IPC changes by the same amount as the actual IPC. Early in the design cycle, achieving good relative accuracy is more important, while, later in the design cycle, absolute accuracy increases in significance since it is important to quantify the exact performance of the processor.

For more in-depth information on additional measurement techniques and simulation methodology, see [50], [10].

4 Recommendations and Future Work

The first half of this section describes three recommendations that add scientific rigor to the simulation process, while the second half describes two avenues for future work.

Recommendation #1: Improve the documentation of simulation methodologies. We examined the proceedings for HPCA, ISCA, and MICRO from 1994 to 2003 for the simulation techniques that were used in each paper and found that we could not determine the specific techniques used in over half of the papers due to inadequate documentation. Also, less a third of the papers in HPCA, ISCA, and MICRO 2001 to 2003 explained why a particular subset of the SPEC CPU 2000 benchmarks was chosen [18]. Obviously, inadequate documentation makes it very difficult for the community to validate those results and build upon that work. Consequently, to ensure repeatability, architects should carefully document their simulation methodology.

Recommendation #2: Computer architects should be careful when choosing a set of parameter values, benchmarks and input sets, and simulation technique. Since poor choices can significantly affect the simulation results, architects should make careful choices and provide justifications for those choices. While this recommendation may seem obvious, we have read many papers that used very unusual processor and memory parameter values. In particular, architects should choose memory latency values that accurately reflect what exists for current-generation processors. Since the memory latency for current-generation processors is a few hundred cycles, choosing a memory latency of 100 cycles for a 16-way issue processor underestimates the penalty of L2 misses. Also, architects should use benchmark classification algorithms found in [27], [28], [132] to help guide their benchmark selections. Finally, based on [133], architects should only use sampling-based reduced-time simulation techniques, e.g., SimPoint, SMARTS, and random, due to their high accuracy.

Recommendation #3: Statistical approaches should be used to help reduce the number of simulations and to analyze the simulation results. Using statistical methods can add scientific rigor to the results, minimize methodological errors, and improve the quality and depth of the work.

Future work on simulation methodology should proceed along two avenues. First, to improve the accuracy of simulation results, more detail should be added to simulators. As described in [46], [9], [24], [25], [38], [14], simulation can accurately predict architectural trends if all components have been modeled in depth. Adding more detail to the simulator to model all components and to model existing components in more depth can only improve the accuracy of the results. For example, adding more detail to accurately model intraprocessor, as well as interprocessor, bus traffic can give more realistic simulation results.

Since increased detailed may result in slower speed, the second avenue of future work is to increase the raw simulator speed or the throughput. Increasing the speed will also allow architects to make more thorough searches of the design space, to simulate more realistic benchmarks (such as OLTP and database workloads), and to simulate a wider range of benchmark programs.

5 Summary

Due to cost, time, and flexibility constraints, simulators are the most important tool available for computer architecture research. However, slow simulation speed and potentially poor accuracy limit the effectiveness of simulators. The causes of insufficient speed are detailed simulators, the large number of benchmarks in a benchmark suite, and the long simulation times of those benchmarks. To reduce the time, architects typically simulate a subset of the benchmarks, use smaller input sets, or simulate only selected pieces of the benchmarks. Unfortunately, these techniques often trade speed for accuracy. Other sources of inaccuracy include poor choices of processor and memory parameter values and poor simulation methodology.

To address these problems, architects have proposed several solutions. To categorize the solutions, previous work in this area was divided into work that focused on:

1. simulator validation and accuracy,
2. processor and memory parameter value selection,
3. benchmark and input set selection,
4. simulation, and
5. performance analysis.

Most previous work focused on the fourth category by proposing solutions that reduced the simulation time while maintaining a similar level of accuracy. More specifically, this paper described and compared techniques, including truncated execution, fast and accurate warmup, reduced input sets, and representative, periodic, and random sampling.

This paper also offers three specific recommendations. Specifically, computer architects should: 1) fully document their simulation steps, 2) carefully choose parameter values, benchmarks, input sets, and techniques, and 3) use statistics to improve the rigor of their work. Adopting these suggestions will help to provide a sound, scientific underpinning for computer architecture research.

References

[1] http://www.futuremark.com/products/3dmark05, 2006.
[2] A. Alameldeen, M. Martin, C. Mauer, K. Moore, M. Xu, D. Sorin, M. Hill, and D. Wood, "Simulating a $2M Commercial Server on a $2K PC," *Computer,* vol. 36, no. 2, pp. 50-57, Feb. 2003.
[3] A. Alameldeen and D. Wood, "Variability in Architectural Simulations of Multi-threaded Workloads," *Proc. Int'l Symp. High Performance Computer Architecture,* 2003.
[4] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C. Tseng, and D. Yeung, "BioBench: A Benchmark Suite of Bioinformatics Applications," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.
[5] http://www.llnl.gov/asci_benchmarks/asci/asci_code_list.html, 2006.
[6] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *Computer,* vol. 35, no. 2, pp. 59-67, Feb. 2002.
[7] K. Barr, H. Pan, M. Zhang, and K. Asanovic, "Accelerating Multiprocessor Simulation with a Memory Timestamp Record," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.
[8] R. Bedichek, "Talisman: Fast and Accurate Multicomputer Simulation," *Proc. Joint Int'l Conf. Measurement and Modeling of Computer Systems,* 1995.
[9] B. Black and J. Shen, "Calibration of Microprocessor Performance Models," *Computer,* vol. 31, no. 5, pp. 59-65, May 1998.
[10] P. Bose and T. Conte, "Performance Analysis and Its Impact on Design," *Computer,* vol. 31 , no. 5, pp. 41-49, May 1998.

[11] P. Bose, T. Conte, and T. Austin, "Challenges in Processor Modeling and Validation," *IEEE Micro,* vol. 19, no. 3, May/June 1999.
[12] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. Int'l Symp. Computer Architecture,* 2000.
[13] D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, P. Emma, and M. Rosenfield, "New Methodology for Early-Stage, Microarchitecture-Level Power-Performance Analysis of Microprocessors," *IBM J. Research and Development,* vol. 47, nos. 5/6, pp. 653-670, Sept. 2003.
[14] H. Cain, K. Lepak, B. Schwartz, and M. Lipasti, "Precise and Accurate Processor Simulation," *Proc. Workshop Computer Architecture Evaluation Using Commercial Workloads,* 2002.
[15] J. Chen, M. Dubois, and P. Stenstrom, "Integrating Complete-System and User-Level Performance/Power Simulators: The SimWattch Approach," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2003.
[16] S. Chen, "Direct SMARTS: Accelerating Microarchitectural Simulation through Direct Execution," master's thesis, Carnegie Mellon Univ., 2004.
[17] M. Chidester and A. George, "Parallel Simulation of Chip-Multiprocessor Architectures," *ACM Trans. Modeling and Computer Simulation,* vol. 12, no. 3, pp. 176-200, July 2002.
[18] D. Citron, "MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences," Panel Discussion at Int'l Symp. Computer Architecture, 2003.
[19] T. Conte, M. Hirsch, and K. Menezes, "Reducing State Loss for Effective Trace Sampling of Superscalar Processors," *Proc. Int'l Conf. Computer Design,* 1996.
[20] T. Conte, M. Hirsch, and W. Hwu, "Combining Trace Sampling with Single Pass Methods for Efficient Cache Simulation," *IEEE Trans. Computers,* vol. 47 , no. 6, pp. 714-720, June 1998.
[21] P. Crowley and J. Baer, "Trace Sampling for Desktop Applications on Windows NT," *Proc. Workshop Workload Characterization,* 1998.
[22] P. Crowley and J. Baer, "On the Use of Trace Sampling for Architectural Studies of Desktop Applications," *Proc. Joint Int'l Conf. Measurement and Modeling of Computer Systems,* 1999.
[23] H. Davis, S. Goldschmidt, and J. Hennessy, "Multiprocessor Simulation and Tracing Using Tango," *Proc. Int'l Conf. Parallel Processing,* 1991.
[24] R. Desikan, D. Burger, and S. Keckler, "Measuring Experimental Error in Microprocessor Simulation," *Proc. Int'l Symp. Computer Architecture,* 2001.
[25] R. Desikan, D. Burger, S. Keckler, L. Cruz, F. Latorre, A. González, and M. Valero, "Errata On: Measuring Experimental Error in Microprocessor Simulation," *Computer Architecture News,* vol. 30, no. 1, Mar. 2002.
[26] M. Durbhakula, V. Pai, and S. Adve, "Improving the Accuracy vs. Speed Tradeoff for Simulating Shared-Memory Multiprocessors with ILP Processors," *Proc. Int'l Symp. High Performance Computer Architecture,* 1999.
[27] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "How Input Data Sets Change Program Behaviour," *Proc. Workshop Computer Architecture Evaluation Using Commercial Workloads,* 2002.
[28] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Workload Design: Selecting Representative Program-Input Pairs," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* 2002.
[29] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Designing Computer Architecture Workloads," *Computer,* vol. 36, no. 2, pp. 65-71, Feb. 2003.
[30] L. Eeckhout, S. Eyerman, B. Callens, and K. De Bosschere, "Accurately Warmed-Up Trace Samples for the Evaluation of Cache Memories," *Proc. High Performance Computing Symp.,* 2003.
[31] L. Eeckhout and K. De Bosschere, "Efficient Simulation of Trace Samples on Parallel Machines," *Parallel Computing,* vol. 30, no. 3, pp. 317-335, Mar. 2004.
[32] L. Eeckhout and K. De Bosschere, "Speeding Up Architectural Simulations for High Performance Processors," *SIMULATION: Trans. Soc. Modeling and Simulation Int'l,* vol. 80, no. 9, pp. 451-468, 2004.
[33] http://www.eembc.org, 2006.
[34] M. Ekman and P. Stenstrom, "Enhancing Multiprocessor Architecture Simulation Speed Using Matched-Pair Comparison," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.
[35] J. Emer, P. Ahuja, E. Borch, A. Klauser, C. Luk, S. Manne, S. Mukherjee, H. Patil, S. Wallace, N. Binkert, R. Espasa, and T. Juan, "Asim: A Performance Model Framework," *Computer,* vol. 35, no. 2, pp. 68-76, Feb. 2002.
[36] B. Falsafi and D. Wood, "Modeling Cost/Performance of a Parallel Computer Simulator," *ACM Trans. Modeling and Computer Simulation,* vol. 7, no. 1, pp. 104-130, Jan. 1997.
[37] http://www.cs.wisc.edu/gems, 2006.
[38] J. Gibson, R. Kunz, D. Ofelt, M. Horowitz, J. Hennessy, and M. Heinrich, "FLASH vs. (Simulated) FLASH: Closing the Simulation Loop," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 2000.
[39] S. Girbal, G. Mouchard, A. Cohen, and O. Temam, "DiST A Simple, Reliable, and Scalable Method to Significantly Reduce Processor Architecture Simulation Time," *Proc. Joint Int'l Conf. Measurement and Modeling of Computer Systems,* 2003.
[40] I. Gómez, L. Pifiuel, M. Prieto, and F. Tirado, "Analysis of Simulation-Adapted SPEC 2000 Benchmarks," *Computer Architecture News,* vol. 30, no. 4, pp. 4-10, Sept. 2002.
[41] J.L. Gustafson and Q.O. Snell, "HINT: A New Way to Measure Computer Performance," *Proc. Hawaii Int'l Conf. System Sciences,* 1995.
[42] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A Free, Commercially Representative Embedded Benchmark Suite," *Proc. Workshop Workload Characterization,* 2001.
[43] J. Haskins Jr. and K. Skadron, "Minimal Subset Evaluation: Rapid Warm-up for Simulated Hardware State," *Proc. Int'l Conf. Computer Design,* 2001.
[44] J. Haskins Jr. and K. Skadron, "Memory Reference Reuse Latency: Accelerated Sampled Microarchitecture Simulation," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2003.
[45] J. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *Computer,* vol. 33, no. 7, pp. 28-35, July 2000.
[46] C. Hughes, V. Pai, P. Ranganathan, and S. Adve, "Rsim: Simulating Shared-Memory Multiprocessors with ILP Processors," *Computer,* vol. 35, no. 2, pp. 40-49, Feb. 2002.
[47] V. Iyengar, L. Trevillyan, and P. Bose, "Representative Traces for Processor Models with Infinite Cache," *Proc. Int'l Symp. High-Performance Computer Architecture,* 1996.
[48] http://www.epcc.ed.ac.uk/javagrande/index_1.html, 2006.
[49] L. Kurian John, "Benchmarks," (draft) *Modern Simulation and Analysis Techniques,* L. Kurian John and L. Eeckhout, eds., CRC Press, to be published.
[50] L. Kurian John, "Performance Modeling and Measurement Techniques," (draft) *Modern Simulation and Analysis Techniques,* L. Kurian John and L. Eeckhout, eds., CRC Press, to be published.
[51] R. Kessler, M. Hill, and D. Wood, "A Comparison of Trace-Sampling Techniques for Multi-Megabyte Caches," *IEEE Trans. Computers,* vol. 43, no. 6, pp. 664-675, June 1994.
[52] H. Khalid, "Validating Trace-Driven Microarchitectural Simulations," *IEEE Micro,* vol. 20, no. 6, pp. 76-82, Nov./Dec. 2000.
[53] A. KleinOsowski and D. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research," *Computer Architecture Letters,* vol. 1, June 2002.
[54] V. Kirshnan and J. Torrellas, "A Direct-Execution Framework for Fast and Accurate Simulation of Superscalar Processor," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* 1998.
[55] T. Lafage and A. Seznec, "Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream," *Proc. Workshop Workload Characterization,* 2000.
[56] S. Laha, J. Patel, and R. Iyer, "Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems," *IEEE Trans. Computers,* vol. 37, no. 11, pp. 1325-1336, Nov. 1988.
[57] E. Larson, S. Chatterjee, and T. Austin, "MASE: A Novel Infrastructure for Detailed Microarchitectural Modeling," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2001.
[58] J. Lau, S. Schoenmackers, and B. Calder, "Structures for Phase Classification," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2004.
[59] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder, "Motivation for Variable Length Intervals and Hierarchical Phase Behavior," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.

[60] G. Lauterbach, "Accelerating Architectural Simulation by Parallel Execution of Trace Samples," Sun Microsystems Laboratory Technical Report TR-93-22, 1993.
[61] B. Lee and L. Kurian John, "NpBench: A Benchmark Suite for Control Plane and Data Plane Applications for Network Processors," *Proc. Int'l Conf. Computer Design,* 2003.
[62] C. Lee, M. Potkonjak, and W. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems," *Proc. Int'l Symp. Microarchitecture,* 1997.
[63] D. Lilja, *Measuring Computer Performance: A Practitioner's Guide.* Cambridge Univ. Press, 2000.
[64] W. Liu and M. Huang, "EXPERT: Expedited Simulation Exploiting Program Behavior Repetition," *Proc. Int'l Conf. Supercomputing,* 2004.
[65] http://www.bitmover.com/lm/lmbench, 2006.
[66] Y. Luo and L. Kurian John, "Efficiently Evaluating Speedup Using Sampled Processor Simulation," *Computer Architecture Letters,* vol. 3, Sept. 2004.
[67] Y. Luo, L. Kurian John, and L. Eeckhout, "Self-Monitored Adaptive Cache Warm-Up for Microprocessor Simulation," *Proc. Symp. Computer Architecture and High Performance Computing,* 2004.
[68] http://m5.eecs.umich.edu, 2006.
[69] P. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Halberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50-58, Feb. 2002.
[70] N. Manjikian, "Multiprocessor Enhancements of the SimpleScalar Tool Set," *Computer Architecture News,* vol. 29, no. 1, pp. 8-15, Mar. 2001.
[71] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset," *Computer Architecture News,* submitted.
[72] J. Mathew, P. Coddington, and K. Hawick, "Analysis and Development of the Java Grande Benchmarks," *Proc. Java Grande Conf.,* 1999.
[73] C. Mauer, M. Hill, and D. Wood, "Full-System Timing-First Simulation," *Proc. SIGMETRICS,* 2002.
[74] G. Memik, W. Mangione-Smith, and W. Hu, "NetBench: A Benchmarking Suite for Network Processors," *Proc. Int'l Conf. Computer Aided Design,* 2001.
[75] G. Memik, W. Mangione-Smith, and W. Hu, "NetBench: A Benchmarking Suite for Network Processors," *Proc. Int'l Conf. Computer-Aided Design (ICCAD),* 2001.
[76] P. Mishra, N. Dutt, and A. Nicolau, "Functional Abstraction Design Space Exploration of Heterogeneous Programmable Architectures," *Proc. Int'l Symp. System Synthesis,* 2001.
[77] M. Moudgill, "Techniques for Fast Simulation of Associative Cache Directories," IBM Research Report RC21038, 1997.
[78] M. Moudgill, "Techniques for Implementing Fast Processor Simulators," *Proc. Ann. Simulation Symp.,* 1998.
[79] M. Moudgill, P. Bose, and J. Moreno, "Validation of Turandot, a Fast Processor Model for Microarchitecture Exploration," *Proc. Int'l Performance, Computing, and Comm. Conf.,* 1999.
[80] S. Murkerjee, S. Reinhardt, B. Falsafi, M. Litzkow, S. Huss-Lederman, M. Hill, J. Larus, and D. Wood, "Fast and Portable Parallel Architecture Simulators: Wisconsin Wind Tunnel II," *IEEE Concurrency,* vol. 8, no. 4, pp. 12-20, Oct.-Dec. 2000.
[81] http://www.nas.nasa.gov/Software/NPB, 2006.
[82] A. Nguyen, M. Michael, A. Sharma, and J. Torrellas, "The Augmint Multiprocessor Simulation Toolkit for Intel x86 Architectures," *Proc. Int'l Conf. Computer Design,* 1996.
[83] A. Nguyen, P. Bose, K. Ekanadham, A. Nanda, and M. Michael, "Accuracy and Speed-Up of Parallel Trace-Driven Architectural Simulation," *Proc. Int'l Parallel Processing Symp.,* 1997.
[84] S. Pees, A. Hoffmann, V. Zivojnovic, and H. Meyr, "LISA—Machine Description Language for Cycle-Accurate Models of Programmable DSP Architectures," *Proc. Design Automation Conf.,* 1999.
[85] D. Penry and D. August, "Optimizations for a Simulator Construction System Supporting Reusable Components," *Proc. Design Automation Conf.,* 2003.
[86] E. Perelman, G. Hamerly, and B. Calder, "Picking Statistically Valid and Early Simulation Points," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* 2003.
[87] D. Perez, G. Mouchard, and O. Temam, "MicroLib: A Case for Quantitative Comparison of Microarchitecture Mechanisms," *Proc. Int'l Symp. Microarchitecture,* 2004.
[88] A. Phansalkar, A. Joshi, L. Eeckhout, and L. John, "Measuring Program Similarity: Experiments with SPEC CPU Benchmark Suites," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.
[89] D. Poulsen and P. Yew, "Execution-Driven Tools for Parallel Simulation of Parallel Architectures and Applications," *Proc. Supercomputing,* 1993.
[90] http://www.eecs.umich.edu/panalyzer, 2006.
[91] http://eda.ee.ucla.edu/PowerImpact, 2006.
[92] J. Ringenberg, C. Pelosi, D. Oehmke, and T. Mudge, "Intrinistic Checkpointing: A Methodology for Decreasing Simulation Time through Binary Modification," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2005.
[93] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta, "Complete Computer Simulation: The SimOS Approach," *Parallel and Distributed Technology,* vol. 3, no. 4, pp. 35-43, Winter 1995.
[94] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod, "Using the SimOS Machine Simulator to Study Complex Computer Systems," *Trans. Modeling and Computer Simulation,* vol. 7, no. 1, pp. 78-103, Jan. 1997.
[95] http://rsim.cs.uiuc.edu/distribution, 2006.
[96] E. Schnarr and J. Larus, "Fast Out-of-Order Processor Simulation Using Memoization," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 1998.
[97] http://math.nist.gov/scimark2, 2006.
[98] H. Shafi, P. Bohrer, J. Phelan, C. Rusu, and J. Peterson, "Design and Validation of a Performance and Power Simulator for PowerPC Systems," *IBM J. Research and Development,* vol. 47, nos. 5/6, pp. 641-651, Sept./Nov. 2003.
[99] T. Sherwood, E. Perelman, and B. Calder, ""Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications," *Proc. Int'l Conf. Parallel Architectures and Compilation Techniques,* 2001.
[100] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems,* 2002.
[101] http://research.compaq.com/wrl/projects/SimOS/SimOs.html, 2004.
[102] http://www.research.ibm.com/arl/projects/SimOSppc.html, 2006.
[103] J. Singh, W. Weber, and A. Gupta, "SPLASH: The Stanford ParalleL Application for SHared Memory," *Computer Architecture News,* vol. 20, no. 1, pp. 5-44, 1992.
[104] K. Skadron, P. Ahuja, M. Martonosi, and D. Clark, "Branch Prediction, Instruction-Window Size, and Cache Size: Performance Trade-Offs and Simulation Techniques," *IEEE Trans. Computers,* vol. 48, no. 11, pp. 1260-1281, Nov. 1999.
[105] http://www.spec.org/benchmarks.html, 2006.
[106] http://www.spec.org/hpg, 2006.
[107] http://www.specbench.org/jbb2000, 2006.
[108] http://www.specbench.org/jvm98, 2006.
[109] http://www.futuremark.com/products/sysmark2004, 2006.
[110] http://www.tpc.org, 2006.
[111] M. Vachharajani, N. Vachharajani, D. Penry, J. Blome, and D. August, "Microarchitectural Exploration with Liberty," *Proc. Int'l Symp. Microarchitecture,* 2002.
[112] M. Vachharajani, N. Vachharajani, D. Penry, J. Blome, and D. August, "The Liberty Simulation Environment, Version 1.0," *Performance Evaluation Review: Special Issue on Tools for Architecture Research,* vol. 31, no. 4, Mar. 2004.
[113] M. Vachharajani, N. Vachharajani, and D. August, "The Liberty Structural Specification Language: A High-Level Modeling Language for Component Reuse," *Proc. Conf. Programming Language Design and Implementation,* 2004.
[114] M. Van Biesbrouck, T. Sherwood, and B. Calder, "A Co-Phase Matrix to Guide Simultaneous Multithreading Simulation," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2004.
[115] M. Van Biesbrouck, L. Eeckhout, and B. Calder, "Efficient Sampling Startup for Sampled Processor Simulation," *Proc. Int'l Conf. High Performance Embedded Architectures and Compilers,* 2005.
[116] H. Vandierendonck and K. De Bosschere, "Eccentric and Fragile Benchmarks," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2004.

[117] H. Vandierendonck and K. De Bosschere, "Experiments with Subsetting Benchmark Suites," *Proc. Workshop Workload Characterization,* 2004.
[118] J. Veenstra and R. Fowler, "MINT: A Front End for Efficient Simulation of Shared-Memory Multiprocessors," *Proc. Int'l Workshop Modeling, Analysis, and Simulation on Computer and Telecomm. Systems,* 1994.
[119] N. Vijaykrishnan, M. Kandemir, M. Irwin, H. Kim, and W. Ye, "Energy-Driven Integrated Hardware-Software Optimizations Using SimplePower," *Proc. Int'l Symp. Computer Architecture,* 2000.
[120] "Introduction to the Simics Full-System Simulator without Equal,"Virtutech White Paper, 2002.
[121] http://www.volano.com/benchmarks.html, 2006.
[122] W. Wang and J. Baer, "Efficient Trace-Driven Simulation Methods for Cache Performance Analysis," *ACM Trans. Computer Systems,* vol. 9, no. 3, pp. 222-241, Aug. 1991.
[123] R. Weicker, "An Overview of Common Benchmarks," *Computer,* vol. 23, no. 12, pp. 65-75, Dec. 1990.
[124] A. Weiss, "The Standardization of Embedded Benchmarking: Pitfalls and Opportunities," *Proc. Int'l Conf. Computer Design,* 1999.
[125] T. Wenisch, R. Wunderlich, B. Falsafi, and J. Hoe, "TurboSMARTS: Accurate Microarchitecture Simulation Sampling in Minutes," Poster at the Int'l Conf. Measurement and Modeling of Computer Systems, 2005.
[126] E. Witchel and M. Rosenblum, "Embra: Fast and Flexible Machine Simulation," *Proc. Joint Conf. Measurement and Modeling of Computer Systems,* 1996.
[127] T. Wolf and M. Franklin, "Commbench—A Telecommunications Benchmark for Network Processors," *Proc. Int'l Symp. Performance Analysis of Systems and Software,* 2000.
[128] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *Proc. Int'l Symp. Computer Architecture,* 1995.
[129] D. Wood, M. Hill, and R. Kessler, "A Model for Estimating Trace-Sample Miss Ratios," *Proc. Conf. Measurement and Modeling of Computer Systems,* 1991.
[130] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "SMARTS: Accelerating Microarchitectural Simulation via Rigorous Statistical Sampling," *Proc. Int'l Symp. Computer Architecture,* 2003.
[131] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "An Evaluation of Stratified Sampling of Microarchitecture Simulations," *Proc. Workshop Duplicating, Deconstructing, and Debunking,* 2004.
[132] J. Yi, D. Lilja, and D. Hawkins, "A Statistically-Rigorous Approach for Improving Simulation Methodology," *Proc. Int'l Symp. High-Performance Computer Architecture,* 2003.
[133] J. Yi, S. Kodakara, R. Sendag, D. Lilja, and D. Hawkins, "Characterizing and Comparing Prevailing Simulation Methodologies," *Proc. Int'l Symp. High-Performance Computer Architecture,* 2005.

Joshua J. Yi received the PhD, MS, and BS degrees, all in electrical engineering, from the University of Minnesota in Minneapolis. He is currently a performance analyst at Freescale Semiconductor, Inc. in Austin, Texas. His research interests include high-performance computer architecture, simulation, benchmarking, low power design, and reliable computing. He is a member of the IEEE and the IEEE Computer Society.

David J. Lilja received the PhD and MS degrees, both in electrical engineering, from the University of Illinois at Urbana-Champaign and the BS degree in computer engineering from Iowa State University. He is currently a professor of electrical and computer engineering and a fellow of the Minnesota Supercomputing Institute at the University of Minnesota in Minneapolis. He also serves as a member of the graduate faculties in computer science and scientific computation. He has been a visiting senior engineer in the Hardware Performance Analysis Group at IBM in Rochester, Minnesota, and a visiting professor at the University of Western Australia in Perth, supported by a Fulbright award. Previously, he worked as a research assistant at the Center for Supercomputing Research and Development at the University of Illinois and as a development engineer at Tandem Computers Inc. (now, a division of Hewlett-Packard) in Cupertino, California. He has chaired and served on the program committees of numerous conferences, was a distinguished visitor of the IEEE Computer Society, is a member of the ACM, the IEEE Computer Society, a fellow of the IEEE, and is a registered professional engineer in electrical engineering in Minnesota and California. His primary research interests are in high-performance computer architecture, parallel computing, hardware-software interactions, nano-computing, and performance analysis.