Techniques for Accurate, Accelerated Processor Simulation: An Analysis of Reduced Inputs and Sampling

John W. Haskins, Jr. Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
{predator,skadron}@cs.virginia.edu

AJ KleinOsowski David J. Lilja
Department of Electrical & Computer Engineering
University of Minnesota
200 Union St. S.E.
Minneapolis, MN 55455
{ajko,lilja}@ece.umn.edu

UVA Computer Science Technical Report CS-2002-01

Abstract

Detailed execution-driven simulation is an important tool for computer architecture research. It is desirable to drive these simulations with standard benchmark programs that are commonly used to evaluate existing computer systems, such as the SPEC2000 suite. Unfortunately, simulating these benchmark programs to completion using full-detail, cycle-accurate simulation on the designated reference input sets results in intractably long simulation durations. This study evaluates and compares two techniques for combating long simulation times: *reduced inputs* and *sampling*. Our objective is to assess the ability of each to reduce simulation running times, while simultaneously minimizing the difference in the results generated by using these techniques relative to the results generated by simulating the benchmark programs to completion using the reference inputs. With the reduced input technique, new input sets are carefully generated by hand to produce run-time characteristics of the benchmark programs that are comparable to the overall characteristics produced when the programs are run with the standard inputs. Sampling, on the other hand, simulates only a small fraction of the program's overall execution in full, cycle-accurate detail using the reference inputs.

1 Introduction

Execution-driven simulation has become the *de facto* standard technique for evaluating the performance of new ideas and mechanisms in the computer architecture research community. With this type of simulation, an application program is compiled, assembled, and executed by another program that models the operations and timing of the new system that the computer architect is attempting to evaluate. The typical output of this type of simulation is a count of the number of cycles that are required to execute the application program, along with a range of additional statistics that help the architect understand how the components of the simulated system behaved to produce this estimated execution time.

The key to making this type of simulation effective in evaluating new ideas is selecting application programs that are representative of the applications that users would be expected to execute on a real implementation of the simulated system. These application programs are called *benchmark* programs since they are used as standard reference points when comparing different architectures [20]. If these benchmark programs are sufficiently representative of the applications that ultimately will be executed on the system being simulated, the resulting simulated results can be used to predict how well the new system will execute the users' application programs.

Detailed studies in the area of processor architecture require *cycle-accurate* simulation that can model the step-by-step flow of instructions through complex processor pipelines. Unfortunately, these simulations are terribly slow, with slowdown factors of hundreds or thousands compared to native execution. With cycle-accurate simulations, running many of the SPEC95 benchmarks to completion with "reference" inputs takes days or weeks [28], and running some of the SPEC2000 benchmarks can take as much as a year [17]. This problem will only get worse as cycle-accurate performance simulators must model ever-more complex futuristic microarchitectures, possibly adding cycle-by-cycle tracking of wire delays, power, temperature, and other metrics of growing interest to the computer architecture community.

Running large benchmarks like SPEC2000 to completion with the reference inputs is clearly not feasible for many types of simulation studies. Analytic modeling techniques [3, 24, 29] and statistical simulation [8, 12, 25, 26] based on typical program characteristics can dramatically speed up parameter-space searches, but there remains a substantial demand for detailed, cycle-accurate simulation of real benchmarks with real inputs. This paper compares two commonly-used techniques: reduced inputs and sampling. Reduced inputs save simulation time by using smaller, yet still realistic inputs. Sampling saves time by performing full-detail, cycle-accurate simulation for one or more samples of the entire program's execution, usually on the reference inputs. We discuss the pros and cons of each approach in terms of ease of use and overall accuracy with respect to the behavior of the original reference input.

Overall, we find that both techniques can substantially reduce the time required to simulate a baseline processor architecture while producing acceptably small errors when compared to simulating the complete benchmark with the reference inputs. The reduced input technique allows the simulation of the entire execution of a benchmark program, including the initialization, computation, and clean-up phases. Sampling, on the other hand, provides a powerful technique for accelerating simulations that use the reference inputs by simulating a subset of the program that is (hopefully) representative of the whole.

The remainder of this paper is organized as follows. Section 2 and Section 3 describe the reduced inputs and sampling techniques, respectively. Section 4 explains our experimental methodology. Section 5 presents our data and discussion, and finally, Section 6 concludes the paper.

2 Reduced Inputs

The benchmark suite from the Standard Performance Evaluation Corporation, commonly known as SPEC [30], is one example of a collection of programs used by the research community to test and rate current and future computer architecture designs [7, 10, 23]. As with most benchmark programs, the SPEC95 benchmark suite was developed with then current and next generation computer systems in mind. Now, six years later, computer technology has advanced to the point where the 1995 benchmarks are no longer suitable; on state-of-the-art computer systems, several of the SPEC95 benchmark programs execute in less than one minute [30]. In an effort to keep up with the rapid progress of computer systems, SPEC chose to dramatically increase the runtimes of the new SPEC2000 benchmark programs [11, 31] as compared to the runtimes of the SPEC95 benchmark programs.

These longer run times are beneficial when testing performance on native hardware. However, when evaluating new computer architectures using detailed execution-driven simulators, the long runtimes of the SPEC2000 benchmarks result in intractably long simulation times.

Reasonable execution times for simulation-based computer architecture research comes in several flavors:

1. Short simulation time (on the order of minutes) to help debug the simulator and do quick tests.

2. Intermediate length simulation time (of a few hours) for more detailed testing of the simulator and to obtain preliminary performance results.

3. A complete simulation (of no more than a few days) using a large, realistic input set to obtain true performance statistics for the experimental architecture design.

Items 1 and 2 do not have to match the execution profile of the original full input set that closely, although it is preferable for item 2 to be reasonably close. For accurate architectural research simulations, however, item 3 should match the profile of the original reference input set to within an acceptable tolerance as measured using an appropriate statistical test.

One could argue that it also does not really matter whether item 3 matches the original profile or not. Rather, it is simply another execution of another input set, which constitutes a valid benchmark in and of itself. Nevertheless, it may be that the original run was carefully chosen by the benchmark developers to display certain characteristics. Therefore, it is highly desirable to have the reduced simulated run match the original characteristics, perhaps within some small margin of error. Furthermore, in the case of SPEC, the original behavior constitutes a standard of sorts among the research community. The goal should not be to create an entirely new standard, but rather to ensure that the characteristics of the new input set are close enough to allow comparisons across research projects.

Using reduced input sets allows a program to run in its entirety, capturing all of the details of the initialization, computation, and clean-up phases. However, small inputs will undoubtedly cause "smaller" behavior. In other words, the program's memory footprint will be smaller. Additionally, the computation phase of the program may be small relative to the initialization and clean-up phases.

A typical approach to reducing simulation runtime is to alter the input dataset. However, blindly reducing a dataset is bad practice since the new input data may cause the execution profile to be completely different from the execution profile obtained with the original reference input dataset. The SPEC committee chose programs for SPEC2000 to obtain a variety of behaviors in terms of how different applications stress hardware components, such as functional units, fetch units, and memory systems. When the execution profile is altered, the benchmark program may no longer test the architecture characteristics it was designed to test.

SPEC provides three standard datasets with their benchmark programs. These data sets are similar to the reduced datasets except on a much larger scale. Namely, the *test* dataset gives a quick test of the benchmark on the desired architecture, the *train* dataset gives an intermediate length run, and the *reference* dataset provides a complete evaluation of the host computer system's performance.

To address the long simulation runtimes of the SPEC2000 programs, the ongoing *SPEClite* project [17] reduces the input datasets of the SPEC2000 programs in a quantitatively defensible way. Along with the small input files, this project has gathered and reported the program profiles and characteristics which result from running the SPEC2000 programs with the newly developed input files. These input files and profiles are available from SPEC free of charge to all SPEC2000 licensees.

The method for creating the small input datasets described in [17] varied widely from benchmark to benchmark. For some benchmarks, the input files were directly truncated or the command line arguments were altered. For benchmarks without input files, and for those with fixed problem parameters, the benchmark source code was examined and modified to alter the number of loop iterations, or some other iteration factor, to reduce the runtime. For still other benchmarks, the benchmark author supplied alternative input data sets.

The SPEClite inputs, when run with a cycle-accurate, execution-driven simulator, run in an average of 4 hours, with the worst case finishing in just less than 6.5 hours.

3 Sampling

A large body of prior work has explored sampling techniques for computer architecture research. For example, Laha, Patel, and Iyer [19]; Crowley and Baer [6]; Martonosi, Gupta, and Anderson [22]; Kaplan, Smaragdakis, and Wilson [14]; and Elnozahy [9] examine memory reference trace sampling and present new algorithms for trace reduction and compression. Other work has studied analytic models for estimating cache miss rates during the unprimed portion of the sample [15, 33], or described means for bounding errors by adjusting simulation lengths [21].

The most widely used sampling technique in the processor architecture community is to perform full-detail simulation for a single, large segment of execution, anywhere from tens of millions to billions of instructions long. When using this approach, the choice of a representative sample becomes critical. Skadron *et al.* showed that the accuracy of this single sample is substantially improved by avoiding potentially unrepresentative initial phases of the program's execution, and can be further improved by careful selection of the sample to capture representative branch-prediction and cache behavior. Unfortunately, the techniques they proposed are ad-hoc. As a better solution, Sherwood, Perelman, and Calder [27] describe how to analyze the distribution of basic-block characteristics to find periodicities in program behavior and accordingly select the sample location to minimize *sampling error*[1].

A more attractive sampling approach would be to use multiple, possibly smaller samples, distributed over the full length of the program's execution. Assuming the location and duration of the samples are well chosen, as the number of samples increases, the sampling error decreases. These samples can be chosen randomly, with an even distribution throughout the program's execution, or with the assistance of metrics that identify how representative a given sample is of the overall program's execution [12, 18].

To make the use of multiple samples feasible, the samples must provide an accurate representation of the program's execution and the samples must execute with an accurate picture of the processor state. This latter issue—of avoiding the so-called *cold-start bias*—is the chief problem. As the fraction of references or instructions modeled becomes smaller, the question of how to "prime" large structures (*i.e.*, how to deal with the unknown state at the beginning of each sample) becomes important. If the state of large structures like caches and the branch predictor do not reflect the execution of the segments between samples, their state will be inaccurate when a new sample begins. The simulated execution therefore, will often be substantially inaccurate because caches and branch predictors have so much leverage over performance. The natural way to avoid cold-start bias is to entirely avoid state loss and model all cache and branch predictor references, even in non-sampled portions of simulation. Unfortunately, this is too expensive: if samples are distributed across the full length of a long program's execution, the cost of moving from the end of one sample to the beginning of the next sample becomes too expensive. Kessler, Hill, and Wood [16] and Conte, Hirsch, Menezes, and Hwu [4, 5] describe various techniques for reducing cold-start bias for cache and branch-predictor simulation.

Unfortunately, these prior techniques for dealing with cold-start bias are heuristics whose accuracy can only be verified experimentally. Haskins and Skadron [13] describe Minimal Subset Evaluation (MSE): a two-pass method that, for a user-specified probability of accuracy, probabilistically determines a minimally

[1]By *sampling error*, we refer to the deviation between quantitative results obtained by full execution relative to results obtained by executing only a subset of the benchmark in cycle-accurate detail

sufficient continguous fraction of the set of non-sampled transactions that must be simulated for warm up to accurately produce state as it would have appeared had the entire fast-forward interval been used for warm up. The end result is a three-stage simulation loop:

1. **Fast-forward.** Rapidly execute instructions, updating only architected state[2].

2. **Warm-up.** At the conclusion of the prescribed fast-forward stage and before full-detail simulation, transition into "warm-up" mode where in addition to updating architected state, all memory and control-flow instructions' transactions with the cache and branch predictor, respectively are modeled.

3. **Full-detail.** Model all processor behavior at the desired, full level of detail, *e.g.*, cycle-accurate pipeline simulation.

This three-stage loop is repeated once for every full-detail sample within the benchmark.

The MSE formulas determine the probability that warming-up only a t-instruction-long contiguous subset of instructions prior to the beginning of a full-detail sample will accurately reproduce large-structure state. The MSE approach consists of the following steps:

1. First, the user chooses the location of simulation sample(s) within the benchmark.

2. The user then selects a desired probability of accuracy $p \in (0, 1)$. This value is used to determine for a given configuration of the cache or other structure, whether the contiguous subset of instructions will with probability p reproduce the simulated hardware state exactly as if that structure's state had been maintained exactly (*i.e.*, had state been modeled for all relavant instructions during the non-sampled segment).

3. Next, the user computes the following formula to obtain the number m of unique references required in each warm-up interval.

$$p = 1 - \frac{\sum_{k=1}^{\lceil \alpha N \rceil - 1} \binom{N}{k} k^m}{\sum_{k=1}^{\lceil \alpha N \rceil} \binom{N}{k} k^m}$$

   This particular formula is for a direct-mapped structure. The numerator is the sum of the number of ways to touch at most $\alpha N - 1$ entries after m unique references. The denominator is the sum of the number of ways to touch at least αN entries after m unique references. For $m \geq \alpha N$ their quotient is the probability of failing to touch αN sets at least once. One minus this quotient is the probability p of succeeding to touch αN entries after m unique references. Iteration finds the value of m required to satisfy the user's specified probability. The parameter α is present to deal with programs that have working sets smaller than an N-entry structure size. For structures with associativity $a > 1$, this formula can be modified to require each set to be touched at least a times.

4. The user profiles the benchmark to characterize, for any point in the benchmark, how many total instructions t must be seen in order to observe m unique references. (By "unique" we mean that among the m references, no two access the same address.) This is a one-time cost for each benchmark–input pair as these profiles are valid for any p, hardware configuration, or sample set of interest.

5. The simulation then enters the three-stage simulation loop described above, aggressively fast-forwarding before entering the MSE-prescribed warm-up stage, before commensing cycle-accurate pipeline simulation.

[2] By *architectued state*, we refer to assembly-language-visable structures such as the register file and main memory

Haskins and Skadron reported errors in IPC of less than 2% for all 15 benchmarks they studied, with a mean error of 0.4%. Their approach reduced total simulation time by at least 30% on all benchmarks and by 60% on average compared to modeling all cache/branch references during non-sample simulation.

4 Experimental Set-up

In order to compare the characteristics of benchmark programs with different input files, we ran simulations using the *sim-outorder* simulator from the SimpleScalar 3.0 suite [1] (compiled for the PISA instruction set) with a subset of the SPEC2000 programs using the reference, train, and SPEClite input sets. For programs with multiple reference command lines, we treated each command line as a separate experiment.

Our experiments with sampled execution on the reference inputs also used *sim-outorder*. We experimented with three different sampling regimes. The first sampling regime uses 50 1-million-instruction samples, the second uses 10 50-million-instruction samples, and the third uses 50 10-million-instruction samples; in all cases, the samples are equidistantly spaced throughout the program. To accelerate movement between samples, we used the MSE technique described in Section 3; we chose our probability of accurate warm-up to be $p = 99.9\%$.

The subset of SPEC2000 programs we chose were written in C and—so that we could simulate their complete cycle-accurate, reference-input executions in a reasonable amount of time—have a dynamic instruction count of less than 200 billion instructions.

From the *sim-outorder* simulation output we extracted and compared the instruction throughput (IPC), branch misprediction rate and level-1 data cache miss rate. These metrics were chosen as representative of the metrics used in most computer architecture studies. For each metric, we use the reference characteristic, X, as our base case and compare the percent error ($100\% \cdot \frac{X_{technique} - X_{reference}}{X_{reference}}$) of the same characteristic for each of our simulation runtime reduction techniques (*i.e.*, different input file, sampling).

Processor Core	
Instruction Window	16-RUU, 8-LSQ
Issue width	4 instructions per cycle
Functional Units	4 IntALU, 1 IntMult/Div, 4 FPALU, 1 FPMult/Div, 2 mem ports
Memory Hierarchy	
L1 D-cache Size	16 KB, 4-way LRU, 32 B blocks
L1 I-cache Size	16 KB, 1-way LRU, 32 B blocks both 1-cycle latency
L2	Unified, 256 KB, 4-way LRU, 64B blocks, 6-cycle latency, WB
Memory	18 cycles
D-TLB Size	128-entry, 4-way LRU, 30-cycle miss penalty
I-TLB Size	64-entry, 4-way LRU, 30-cycle miss penalty
Branch Predictor	
Branch predictor	bimodal (plain 2-bit ctr.), 2 K-entry PHT
Branch target buffer	2 K-entry, 4-way
Return-address-stack	8-entry
Minimum misprediction latency	5 cycles

Table 1: Configuration of simulated processor microarchitecture.

As a processor configuration, we used the *sim-outorder* default configuration, summarized in Table 1. This configuration is not representative of any particular modern processor, but neither are any of the parameters truly outrageous. Benchmarks were compiled for SimpleScalar's MIPS-like portable instruction set architecture (PISA) [2] using gcc 2.6.3 and the -O3 optimization flag. The statically-linked binaries include all library code.

This configuration information is only presented for completeness. We are only exploring how well reduced inputs and sampling replicate the accuracy of full-length simulation, so the choice of processor model should not qualitatively change the results of this paper because all simulations use the same configuration.

5 Evaluation

Tables 2–5 present the data for the SPEClite reduced inputs, giving the percent difference in IPC, first-level data-cache (L1 D-cache) miss rate, and branch misprediction rate as well as running time. As a basis for comparison, these tables also include data for the SPEC-provided "train" inputs.

Tables 6–9 present similar data for the multiple-sample approach. We looked at three sampling regimes: 50 samples of 1 million instructions each (50×10^6), 10 samples of 50 million instructions each ($10 \times 5(10^6)$), and 50 samples of 10 million instructions each (50×10^7)—all equidistantly spaced throughout the benchmark. Data for all three regimes appears in Table 6; subsequent tables just use the 50×10^6 data.

5.1 Performance of Reduced Inputs and Sampling

Both methods have measurable variations in IPC with the benchmark program. The sources of these variations seem to be primarily due to variations in the branch misprediction rate.

For individual applications, we studied the variation among full-detail simulations of the different reference inputs, reported in Table 10. These "cross-input" variations are substantial for *gzip* and especially for *vpr* in terms of instruction throughput. The variations between the reduced inputs and their corresponding reference inputs are within the same range as these cross-input variations for *gzip* and for one of the *vpr* inputs.

The percent difference for cache miss rate for the reduced inputs (as compared to the reference run) was within the same range as the percent variation across different reference-input command lines for all benchmarks except for *art*. The percent error for the branch misprediction rate for the reduced inputs (as compared to the reference-input), however, was far worse than the percent variation across different reference-input command lines. For the sampling approach, the percent error for the cache miss rate was within the same range as the percent variation across different reference-input runs. The percent error for the branch misprediction rate for the sampling approach was within the same range as the percent variation across different reference-input command lines for all benchmarks except *art*.

Our evidence indicates that insofar as IPC is concerned, sampling error is reduced by increasing the number of samples more than increasing the size of individual samples. Table 6 shows that the IPC percent-difference between the sampled input regimes is greatest on most benchmarks for the $10 \times 5(10^6)$ regime which executes the same number of instructions (50 million) in cycle-accurate detail as 50×10^6. The results from the 50×10^7 regime seem counter to intuition in that executing an even greater number of instructions (500 million), spread out over a large number (50) of samples does not yield more accurate results than 50×10^6. Although we have not yet been able to verify this hypothesis, it seems that there exists a "sweet-spot" for sample sizes, beyond which, accuracy no longer improves or even decreases.

benchmark(input)	TRAIN	SPEClite	FULL-DETAIL REF
art (startx=110)	0.5485(-8.34%)	0.5440(-6.68%)	0.5984
art (startx=470)	0.5485(-8.19%)	0.5440(-8.94%)	0.5974
gzip (graphic)	1.4195(3.96%)	1.4171(3.79%)	1.3654
gzip (log)	1.4195(-2.91%)	1.5778(7.91%)	1.4621
gzip (program)	1.4195(2.25%)	1.3671(-1.52%)	1.3882
gzip (random)	1.4195(8.76%)	1.3211(0.18%)	1.3187
gzip (source)	1.4195(4.28%)	1.3765(1.06%)	1.3612
vortex (lendian1)	1.1646(-5.66%)	1.0517(-14.81%)	1.2345
vortex (lendian2)	1.1646(-3.20%)	1.0517(-12.58%)	1.2031
vortex (lendian3)	1.1646(-5.46%)	1.0517(-14.62%)	1.2318
vpr (place)	0.9667(14.54%)	0.9404(11.42%)	0.8440
vpr (route)	1.1450(13.09%)	1.4102(39.28%)	1.0125

Table 2: IPC percent-difference summary for SPEClite simulations.

benchmark(input)	TRAIN	SPEClite	FULL-DETAIL REF
art (startx=110)	0.4777(-4.17%)	0.4495(-9.83%)	0.4985
art (startx=470)	0.4777(-4.33%)	0.4495(-9.97%)	0.4993
gzip (graphic)	0.0640(52.38%)	0.0501(19.29%)	0.0420
gzip (log)	0.0640(-30.89%)	0.0763(-17.60%)	0.0926
gzip (program)	0.0640(-62.29%)	0.1887(11.20%)	0.1697
gzip (random)	0.0640(10.73%)	0.0575(-0.52%)	0.0578
gzip (source)	0.0640(-53.45%)	0.1155(-16.00%)	0.1375
vortex (lendian1)	0.0120(-35.83%)	0.0176(-5.88%)	0.0187
vortex (lendian2)	0.0120(-11.11%)	0.0176(30.37%)	0.0135
vortex (lendian3)	0.0120(-38.46%)	0.0176(-9.74%)	0.0195
vpr (place)	0.0562(-12.73%)	0.0375(-41.77%)	0.0644
vpr (route)	0.0444(-4.52%)	0.0414(-10.97%)	0.0465

Table 3: D-L1 miss-rate percent-difference summary for SPEClite simulations.

benchmark(input)	TRAIN	SPEClite	FULL-DETAIL REF
art (startx=110)	0.0928(76.09%)	0.1752(232.45%)	0.0527
art (startx=470)	0.0928(76.76%)	0.1752(233.71%)	0.0525
gzip (graphic)	0.0737(-16.06%)	0.0783(-10.82%)	0.0878
gzip (log)	0.0737(50.10%)	0.0714(-24.24%)	0.0491
gzip (program)	0.0737(37.76%)	0.0545(1.87%)	0.0535
gzip (random)	0.0737(-27.96%)	0.1017(-0.59%)	0.1023
gzip (source)	0.0737(24.28%)	0.0652(9.95%)	0.0593
vortex (lendian1)	0.0172(56.36%)	0.0201(82.73%)	0.0110
vortex (lendian2)	0.0172(13.91%)	0.0201(33.11%)	0.0151
vortex (lendian3)	0.0172(59.26%)	0.0201(86.11%)	0.0108
vpr (place)	0.0924(1.43%)	0.1195(31.17%)	0.0911
vpr (route)	0.0984(-3.91%)	0.0921(-10.06%)	0.1024

Table 4: B-pred miss-rate percent-difference summary for SPEClite simulations.

benchmark(input)	train	SPEClite	FULL-DETAIL REF
art (startx=110)	89349(11.81%)	23169(3.06%)	756507
art (startx=470)	89349(11.14%)	23169(2.89%)	801815
gzip (graphic)	217809(26.27%)	14381(1.73%)	829097
gzip (log)	217809(57.25%)	6148(1.62%)	380475
gzip (program)	217809(22.85%)	16761(1.76%)	953154
gzip (random)	217809(29.69%)	12965(1.77%)	733661
gzip (source)	217809(32.33%)	13163(1.95%)	673621
vortex (lendian1)	89760(10.08%)	13277(1.49%)	890519
vortex (lendian2)	89760(9.24%)	13277(1.37%)	971445
vortex (lendian3)	89760(9.40%)	13277(1.39%)	954585
vpr (place)	77088(7.34%)	16044(1.53%)	1050041
vpr (route)	57576(5.77%)	10389(1.04%)	997469

Table 5: Wall-clock running time (in seconds) summary for SPEClite simulations.

benchmark (input)	50×10^6	$10 \times 5(10^6)$	50×10^7	FULL-DETAIL REF
art (startx=110)	0.5794(-3.18%)	0.6062(1.30%)	0.5957(-0.45%)	0.5984
art (startx=470)	0.5939(-0.59%)	0.5954(-0.33%)	0.6012(0.64%)	0.5974
gzip (graphic)	1.3483(-1.25%)	1.3317(-2.47%)	1.3414(-1.76%)	1.3654
gzip (log)	1.4521(-0.68%)	1.4259(-2.48%)	1.4414(-1.42%)	1.4621
gzip (program)	1.4190(2.22%)	1.4103(1.59%)	1.3945(0.45%)	1.3882
gzip (random)	1.3267(0.61%)	1.3143(-0.33%)	1.3370(1.39%)	1.3187
gzip (source)	1.4064(3.32%)	1.4209(4.39%)	1.3706(0.69%)	1.3612
vortex (lendian1)	1.0661(-2.35%)	1.1356(4.01%)	1.0944(2.38%)	1.0918
vortex (lendian2)	1.0356(–%)	1.0193(–%)	1.0518(–%)	–
vortex (lendian3)	1.0683(-1.90%)	1.1727(7.69%)	1.0826(-0.59%)	1.0890
vpr (place)	0.8479(0.46%)	0.8436(-0.05%)	0.8502(0.73%)	0.8440
vpr (route)	1.0386(2.58%)	1.0629(4.98%)	1.0409(2.80%)	1.0125

Table 6: IPC percent-difference summary for multiple-sample simulations. NOTICE TO THE REVIEWERS: Elements marked "–" denote data from simulations that had yet completed by the deadline. We were forced to re-run the *vortex* simulations to correct the accidental use of different binary files between experiments. We will integrate these figures into the final draft of the paper if accepted.

benchmark (input)	50×10^6	FULL-DETAIL REF
art (startx=110)	0.4988(0.06%)	0.4985
art (startx=470)	0.4985(-0.16%)	0.4993
gzip (graphic)	0.0427(1.67%)	0.0420
gzip (log)	0.0980(5.83%)	0.0926
gzip (program)	0.1741(2.59%)	0.1697
gzip (random)	0.0599(3.63%)	0.0578
gzip (source)	0.1346(-2.11%)	0.1375
vortex (lendian1)	0.0148(17.46%)	0.0126
vortex (lendian2)	0.0107(–%)	–
vortex (lendian3)	0.0156(19.08%)	0.0131
vpr (place)	0.0636(-1.24%)	0.0644
vpr (route)	0.0459(-1.29%)	0.0465

Table 7: D-L1 miss-rate percent-difference summary for the 50×10^6 multiple-sample simulations.

benchmark (input)	50×10^6	FULL-DETAIL REF
art (startx=110)	0.0523(-0.75%)	0.0527
art (startx=470)	0.0532(1.33%)	0.0525
gzip (graphic)	0.0960(9.34%)	0.0878
gzip (log)	0.0490(-0.20%)	0.0491
gzip (program)	0.0519(-2.99%)	0.0535
gzip (random)	0.0906(-11.44%)	0.1023
gzip (source)	0.0590(-0.51%)	0.0593
vortex (lendian1)	0.0905(-1.20%)	0.0916
vortex (lendian2)	0.1108(–%)	–
vortex (lendian3)	0.0913(-0.11%)	0.0914
vpr (place)	0.0904(-0.77%)	0.0911
vpr (route)	0.1018(11.75%)	0.1024

Table 8: B-pred miss-rate percent-difference summary for the 50×10^6 multiple-sample simulations.

benchmark (input)	50×10^6	FULL-DETAIL REF
art (startx=110)	10319(1.36%)	756507
art (startx=470)	11190(1.40%)	801815
gzip (graphic)	4063(0.49%)	829097
gzip (log)	4711(1.24%)	380475
gzip (program)	12988(1.36%)	953154
gzip (random)	8607(1.17%)	733661
gzip (source)	7766(1.15%)	673621
vortex (lendian1)	27084(6.75%)	401396
vortex (lendian2)	30908(–%)	–
vortex (lendian3)	30111(6.72%)	448293
vpr (place)	15410(1.47%)	1050041
vpr (route)	11734(1.18%)	997469

Table 9: Wall-clock running time (in seconds) summary for the 50×10^6 multiple-sample simulations.

benchmark	B-pred miss rate	D-L1 miss rate	IPC
art	0.38%	0.16%	0.17%
gzip	108.35%	304.05%	10.74%
vortex	39.81%	44.44%	2.61%
vpr	12.40%	38.49%	19.96%

Table 10: Maximum percent difference variation among the multiple full-detail reference-input command lines for each benchmark.

We also see how difficult it is to develop input sets with reduced execution time that still accurately replicate the behavior of the original input. Although the reduced inputs mostly exhibit significantly less error than the training inputs, the variations are still substantial for some benchmarks despite extensive efforts to obtain as close a match as possible.

On the other hand, the reduced inputs are valid inputs in their own right, and simulations using the reduced inputs can be trusted to represent the complete behavior of an actual application with a real input[3], without the need to worry about sampling error of any kind nor the need to develop a sampling methodology.

Still, for an arbitrary application and input, sampling is easier to apply. For a simulator already constructed to perform sampling, the benchmark can be executed immediately if warm-up is used between samples to avoid cold-start bias. If faster warm-up times are required, a profile must be generated first in order to apply the MSE technique. This is still easier and requires minimal human effort compared to the sometimes very difficult task of finding a way to generate a reduced input (if that input is supposed to match something else).

A final consideration is that sampling, if done with a fixed number of fixed-length samples for all benchmarks (as we do here) simplifies the computation of mean results, since arithmetic means can be used. The use of an arithmetic mean is more questionable for comparing ratios (*e.g.*, IPC, miss rate) across workloads of different size, and alternative means are less intuitive to use.

5.2 Simulations Across Host Architectures

In order to perform computer architecture studies in a timely manner, researchers typically increase the throughput of their simulations by running several simulations simultaneously on a pool of available machines. This pool of machines often is not homogeneous, however. For example, the Condor system [32] distributes jobs to whichever networked machines are available.

Table 11 shows the resulting IPC of the four SPEC2000 programs used in our experiments when run on various host architectures. From this table, we see at most single-digit percent differences among all the big-endian architectures used to perform the simulations. When we compare the IPC obtained on the big-endian architectures to that obtained on the little-endian architecture (Linux/x86), we again see very little change. The single exception occurs for the *gzip* program, in which we see an 11 percent difference from Linux/x86 compared to all of the big-endian architectures. Other than this isolated case, we see single digit percent differences across the various host architectures.

Aside from the host machine endianness, differences in architecture and operating system have very little

[3]Indeed, the SPEC inputs may have been artificially inflated in size to obtain long running times.

	little-endian	big-endian		
benchmark	Linux/x86	Solaris/SPARC	AIX/Power3	Irix/MIPS
art	0.5440	0.5436(0.07%)	0.5436(0.07%)	0.5436(0.07%)
gzip (graphic)	1.4171	1.6001(-11.44%)	1.6000(-11.44%)	1.6003(-11.45%)
vortex	1.0517	1.0365(1.47%)	1.0371(1.41%)	1.0352(1.59%)
vpr (place)	0.9404	0.9342(0.68%)	0.9341(0.68%)	0.9341(0.68%)

Table 11: Platform variation IPC percent-difference summary.

impact on the simulation results. Thus, to ensure consistent simulation results, researchers should strive to maintain a homogeneous endianness in the pool of machines used for their simulations.

6 Conclusions

Cycle-accurate execution-driven simulation of new processor designs is an important tool for evaluating and comparing the performance of new architectural ideas. These types of simulations are driven by standard benchmark programs that have been selected to produce certain characteristics and behaviors. The main problem with using existing benchmark programs is the very long wall-clock times that are required to execute the benchmark programs with a given simulator. This study has evaluated and compared two techniques that can be used to reduce the required simulation time. The first technique carefully adjusts the input data set of a benchmark program to reduce the amount of time that it executes. The second technique samples the execution of the benchmark program and simulates only those samples in full, cycle-accurate detail.

Our results show that both the reduced input simulations and the sampling technique can produce significant errors in important program characteristics compared to the characteristics produced when executing the original, unaltered program. Both of these techniques significantly reduce the wall-clock time required to execute the simulations, however.

We find that using carefully reduced input sets is a reasonable and appropriate technique for reducing the simulation time. While the program characteristics produced when executing with these reduced input sets may not exactly match the characteristics of the complete benchmark execution, they do constitute valid inputs in themselves. It useful to know how close important characteristics of the reduced executions are to the original executions, but an exact match is not required to draw conclusions that are still valid for the reduced input. When using reduced input sets, however, it is important for the experimenter to understand and disclose the characteristic profile of the program with the small input sets. Alternatively, it is probably easier to use a pseudo-standard set of reduced input data sets, such as the SPEClite [17] input sets discussed in this study.

Sampling is an appropriate choice for reducing the simulation time of a program for which the researcher must adhrere to a fixed input set. The sampling regimes and sampling acceleration technique (MSE) discussed in this paper can be applied to any program and are shown to have acceptable margins of error. However, the sampling approach used in this paper has the problem of non-monotonic behavior. That is, increasing the number of samples or the length of the samples does not necessarily produce more accurate results. Further analysis using different sample selection techniques (*e.g.*, [27]) is a topic for further study.

We conclude that the two approaches evaluated in this study each boasts its own uses and benefits. The primary advantage of reduced input data sets compared to a sampling technique is that the reduced input technique allows the simluation of the entire execution of a benchmark program, including the initialization, computation, and clean-up phases. Thus, reduced inputs are critical for research that studies the full, end-to-end execution of a benchmark program. Sampling, on the other hand, provides a method for accelerating processor simulations when the input dataset is fixed. Furthermore, sampling provides abundant flexibility, permitting a wide range of sampling strategies—in terms of sample location and duration—to be implemented.

References

[1] T. M. Austin. SimpleScalar home page. http://www.simplescalar.org.

[2] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[3] R. Carl and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the Workshop on Performance Analysis and its Impact on Design*, June 1998.

[4] T. M. Conte, M. A. Hirsch, and W. W. Hwu. Combining trace sampling with single pass methods for efficient cache simulation. *IEEE Transactions on Computers*, 47(6):714–19, June 1998.

[5] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *Proceedings of the 1996 International Conference on Computer Design*, Oct. 1996.

[6] P. Crowley and J.-L. Baer. On the use of trace sampling for architectural studies of desktop applications. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 208–09, May 1999.

[7] J. Dujmovic and I. Dujmovic. Evolution and Evalutaion of SPEC Benchmarks. *Performance Evaluation Review*, 26(3):2–9, Dec. 1998.

[8] L. Eeckhout and K. De Bosschere. Hybrid analytical-statistical modeling for efficiently exploring architecture and workload design spaces. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 25–34, Sept. 2001.

[9] E. N. Elnozahy. Address trace compression through loop detection and reduction. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 214–15, May 1999.

[10] R. Giladi and N. Ahituv. SPEC as a Performance Evaluation Measure. *IEEE Computer*, 28(8):33–42, Aug. 1995.

[11] J.L. Henning. SPEC CPU2000: Measuring CPU Performance in the New Millenium. *IEEE Computer*, 33(7):28–35, July 2000.

[12] V. S. Iyengar and L. H. Trevillyan. Evaluation and generation of reduced traces for benchmarks. Research Report RC 20610, IBM, Oct. 1996.

[13] J. W. Haskins, Jr. and K. Skadron. Minimal subset evaluation: Rapid warm-up for simulated hardware state. In *Proceedings of the 2001 International Conference on Computer Design*, Sept. 2001.

[14] S. F. Kaplan, Y. Smaragdakis, and P. R. Wilson. Trace reduction for virtual memory simulations. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 47–58, May 1999.

[15] R. E. Kessler, Mark D. Hill, and David A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. Tech. Report 1048, University of Wisconsin-Madison Computer Sciences Department, Sep. 1991.

[16] R. E. Kessler, Mark D. Hill, and David A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–75, June 1994.

[17] AJ KleinOsowski, J. Flynn, N. Meares, and D. J. Lilja. Adapting the SPEC 2000 benchmark sutie for simulation-based computer architecture research. In *Proceedings of the Third IEEE Annual Workshop on Workload Characterization*, pages 73–82, Sep. 2000.

[18] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *Proceedings of the Third IEEE Annual Workshop on Workload Characterization*, pages 102–110, Sep. 2000.

[19] Subhasis Laha, Janak H. Patel, and Ravishankar K. Iyer. Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems. *IEEE Transactions on Computers*, 37(11):1325–1336, Nov. 1988.

[20] David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, Cambridge, United Kingdom, 2000.

[21] Margaret Martonosi, Anoop Gupta, and Thomas Anderson. Effectiveness of Trace Sampling for Performance Debugging Tools. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 248–59, May 1993.

[22] Margaret Martonosi, Anoop Gupta, and Thomas Anderson. Tuning Memory Performance in Sequential and Parallel Programs. *IEEE Computer*, pages 32–40, Apr. 1995.

[23] M. Jacoby N. Mirghafori and D. Patterson. Truth in SPEC Benchmarks. *ACM Computer Architecture News*, 23(5):34–42, Dec. 1995.

[24] D. B. Noonburg and J. P. Shen. A framework for statistical modeling of superscalar processor performance. In *Proceedings of the Third International Symposium on High-Performance Computer Architecture*, pages 298–309, Feb. 1997.

[25] S. Nussbaum and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 15–24, Sept. 2001.

[26] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor designs. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 71–82, June 2000.

[27] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pages 3–14, Sept. 2001.

[28] K. Skadron, P. S. Ahuja, M. Martonosi, and D. W. Clark. Branch prediction, instruction-window size, and cache size: Performance tradeoffs and simulation techniques. *IEEE Transactions on Computers*, 48(11):1260–81, Nov. 1999.

[29] D. Sorin, V. Pai, S. Adve, M. Vernon, and D. Wood. Analytic evaluation of shared-memory systems with ILP processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 380–91, June 1998.

[30] Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmarks. http://www.specbench.org/osg/cpu2000.

[31] Standard Performance Evaluation Corporation. SPEC CPU2000 Press Release FAQ. http://www.spec.org/osg/cpu2000/press.faq.html.

[32] University of Wisconsin at Madison. Condor High Throughput Computing Software. http://www.cs.wisc.edu/condor/.

[33] David A. Wood, Mark D. Hill, and R. E. Kessler. A Model for Estimating Trace-Sample Miss Ratios. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 79–89, June 1991.