Computer Architecture Lab at Carnegie Mellon (CALCM) Technical Report 2003-1

Applying SMARTS to SPEC CPU2000[1]

Thomas F. Wenisch Roland E. Wunderlich Babak Falsafi James C. Hoe
Computer Architecture Laboratory (CALCM)
Carnegie Mellon University, Pittsburgh, PA 15213-3890
{rolandw, twenisch, babak, jhoe}@ece.cmu.edu
http://www.ece.cmu.edu/~simflex

Abstract

Current software-based microarchitecture simulators are many orders of magnitude slower than the hardware they simulate. Hence, most microarchitecture design studies draw their conclusions from drastically truncated benchmark simulations that are often inaccurate and misleading. This paper presents the Sampling Microarchitecture Simulation (SMARTS) *framework as an approach to enable fast and accurate performance measurements of full-length benchmarks. SMARTS accelerates simulation by selectively measuring in detail only an appropriate benchmark subset. SMARTS prescribes a statistically sound procedure for configuring a systematic sampling simulation run to achieve a desired quantifiable confidence in estimates.*

Analysis of 41 of the 45 possible SPEC2K benchmark/input combinations show CPI and energy per instruction (EPI) can be estimated to within ±3% with 99.7% confidence by measuring fewer than 50 million instructions per benchmark. In practice, inaccuracy in micro-architectural state initialization introduces an additional uncertainty which we empirically bound to ~2% for the tested benchmarks. Our implementation of SMARTS achieves an actual average error of only 0.64% on CPI and 0.59% on EPI for the tested benchmarks, running with average speedups of 35 and 60 over detailed simulation of 8-way and 16-way out-of-order processors, respectively.

1. Introduction

Computer architects have long relied on software simulation to study the functionality and performance of proposed hardware designs. Despite phenomenal improvement in processor performance over the last decades, the disproportionate growth in hardware complexity that needs to be modeled has steadily eroded simulation speed. Today, the fastest cycle-accurate modern microprocessor performance simulators are more than five orders of magnitude slower than the hardware they model—simulating at a nominal rate of 0.5 MIPS on a 2 GHz Pentium 4. More detailed simulators and register-transfer-level simulators are easily six or more orders of magnitude slower than the proposed hardware. One minute of execution in real time can correspond to days, if not weeks, of simulation time.

1.1. Current approaches

To mitigate prohibitively slow simulation speeds, researchers often use abbreviated instruction execution streams of benchmarks as representative workloads in design studies. More than half of the recent papers in top-tier computer architecture conferences presented performance claims extrapolated from abbreviated runs.[2] Researchers predominantly skip the initial 250 million to two billion instructions and then measure a single section of 100 million to one billion instructions. Unfortunately, several studies [4,10,12,17] have concluded that results based only on a single abbreviated execution stream are inaccurate or misleading because they fail to capture global variations in program behavior and performance.

Another common approach to curtail simulation time is to use fewer or smaller input sets (i.e., the test or train sets rather than all of the reference sets in SPEC2K). Recent papers, however, have also shown benchmark behavior varies significantly across test, train and reference inputs for a number of SPEC2K benchmarks [8,17].

To obtain performance results based on complete benchmarks and input sets, many proposals have advocated statistical [4,7,11,12] or profile-driven [10,17] simulation sampling. Simulation sampling measures only chosen sections (called sampling units) from a benchmark's full execution stream. The sections in between sampling units are "fast-forwarded" using functional simulation that only

1. This technical report contains the complete set of results, but the largely the same text as the SMARTS paper in ISCA 2003. The additional data is presented in the appendices at the end of the report.

2. This past year, 64 papers presented at ISCA, MICRO, and HPCA include simulation results; 38 used a single sampling unit, 20 used reduced input sets or microbenchmarks, and 6 used other approaches.

maintains programmer-visible architectural state. We faced two key challenges to simulation sampling: (1) choosing an appropriate subset with the minimum number of instructions to meet a given error bound, and (2) reconstructing an accurate microarchitectural state (e.g., branch predictor and cache hierarchy contents) for unbiased sample measurement following an extended period of functional fast-forwarding.

Current proposals for simulation sampling suffer from several key shortcomings. On the efficiency front, most proposals sample several orders of magnitude more instructions than are statistically necessary for their stated error [7,10,11,12,17]. This inefficiency is often rooted in their excessively large sampling units, either to amortize the overhead of reconstructing microarchitectural state or to capture coarse-grain performance variations by brute force. On the accuracy front, most proposals either do not offer tight error bounds on their performance estimations [10,11,12,17], or require unrealistic assumptions about the microarchitecture (e.g., perfect branch prediction or cache hierarchies) [4].

1.2. The SMARTS approach

We propose the *Sampling Microarchitecture Simulation* (*SMARTS*) framework which applies statistical sampling theory to address the aforementioned issues in simulation sampling. Unlike prior approaches to simulation sampling, SMARTS prescribes an exact and constructive procedure for selecting a minimal subset from a benchmark's instruction execution stream to achieve a desired confidence interval. SMARTS uses a measure of variability (coefficient of variation) to determine the optimal sample that captures a program's inherent variation. An optimal sample generally consists of a large number of small sampling units. Unbiased measurement of sampling units as small as 1000 instructions is possible by applying careful *functional warming*—maintaining large microarchitectural state, such as branch predictors and the cache hierarchy—during fast-forwarding between sampling units.

We evaluate SMARTS in the context of a wide-issue out-of-order superscalar simulator called SMARTSim which is based on SimpleScalar 3.0 [2]. We employed SMARTSim to estimate the CPI and energy per instruction (EPI) for 41 out of 45 SPEC2K benchmark/input combinations on two microarchitecture configurations. We make the following primary contributions:

- **Optimal sampling:** SMARTSim achieves an actual average error of only 0.64% on CPI and 0.59% on EPI by simulating fewer than 50 million instructions in detail for each of the 41 SPEC2K benchmarks. This represents an exceedingly small fraction of the complete benchmark streams, which range between 2 and 547 billion instructions.
- **Simulation speedup:** On a 2 GHz Pentium 4, SMARTSim can achieve average speedups of 35 and 60 relative to `sim-outorder` for 8-way and 16-way superscalar processor models, respectively. At current processor speeds, these speedups enable simulation speeds of over 9 MIPS.
- **Future impact:** SMARTS sampling simulation rate is, for all practical purposes, decoupled from the speed of the detailed simulator. This result has fundamental bearings on future simulator designs. First, designers should focus less on elaborate performance shortcuts in detailed simulators, and more on increasing the detailed simulator's overall design flexibility and accuracy. Second, designers should focus on developing techniques which speed up fast-forwarding and functional warming (e.g., direct execution [16]), as these ultimately determine sampling simulation rate.

Paper outline. The rest of this paper is organized as follows. Section 2 presents background on statistical simulation. Section 3 presents the SMARTS framework. Section 4 presents an implementation of SMARTS in the context of a microarchitecture simulation infrastructure. Section 5 evaluates the effectiveness of the SMARTS framework at accelerating microarchitecture simulation. Finally, we conclude in Section 6.

2. Statistical sampling

The field of inferential statistics offers well-defined procedures to quantify and to ensure the quality of sample-derived estimates. This section provides basic background on statistical sampling. We describe procedures for selecting a sample for mean estimation and the mathematics for calculating the confidence in an estimate.

Statistical sampling attempts to estimate a given cumulative property of a *population* by measuring only a *sample*, a subset of the population [13]. By examining an appropriately selected sample, one can infer the nature of the property over the whole population in terms of total, mean, and proportion. The theory of sampling is concerned with choosing a minimal but representative sample to achieve a quantifiable accuracy and precision in the estimate. The theory does not presume a normally-distributed population. Our goal is to apply this theory to: (1) identify a minimal but representative sample from the population for microarchitecture simulation, and (2) establish a confidence level for the error on sample estimates.

Table 1 summarizes the standard statistical sampling variables and terminology relevant to this paper. *Simple random sampling* selects a sample of n elements (a.k.a. sampling units) at random from a population of N

Table 1. Sampling variables.

Population variables		Sample variables	
N	size	n	size
$\bar{X}$	mean	$\bar{x}$	mean
σ_x	standard deviation	$\hat{V}_x$	coeff. of variation
V_x	coeff. of variation	$(1-\alpha)$	confidence level
		$\pm\varepsilon \cdot \bar{X}$	confidence interval
		k	systematic-sampling interval
		$B(\bar{x})$	bias of sample mean

elements. Measurements are taken on the selected sampling units, and for a sufficiently large sample size (i.e., $n > 30$) the sampled results can be meaningfully extrapolated to provide an estimate for the whole population. In particular, the true population mean $\bar{X}$ of a property χ is estimated by the sample mean $\bar{x}$. The coefficient of variation is the standard deviation of χ normalized by $\bar{X}$, $V_x = \sigma_x/\bar{X}$. The likelihood that $\bar{x}$ is a good estimate of $\bar{X}$ improves with sample size and decreases with V_x. SMARTS leverages the relationship between n, V_x and desired confidence to minimize the required sample size for a benchmark.

Formally, the confidence in a mean estimate is jointly quantified by two interdependent terms: confidence level $(1-\alpha)$ and confidence interval $\pm\varepsilon \cdot \bar{X}$. The interpretation of confidence level and interval is that, over a large number of random sampling trials, a $(1-\alpha)$ fraction of the trials should produce $\bar{x}$ that is within $\pm\varepsilon \cdot \bar{X}$ of $\bar{X}$.[1] The confidence interval achieved by a sample is $\pm([(z \cdot V_x)/\sqrt{n}] \cdot \bar{X})$ where z is the $100[1-(\alpha/2)]$ percentile of the standard normal distribution. (We assume $N \gg n \gg 1$ to simplify the expressions in this paper.) For a sample with a given V_x and size n, one can choose a desired confidence level and solve for the achieved confidence interval, or vice versa.

To design a sampling simulation to meet a certain confidence, one begins by determining an appropriate n based on the required confidence and V_x, using the same equations above. (Note that the population size does not impact the determination of n.) The true coefficient of variation of a population is rarely available in practice unless the entire population is examined. Instead, $\hat{V}_x$ of a sufficiently large initial sample is commonly used in place of V_x in computing the confidence of that sample. If the initial sample does *not* achieve the desired confidence, the required size of a subsequent sample can be computed using $\hat{V}_x$, where $n \geq ((z \cdot \hat{V}_x)/\varepsilon)^2$. In practice, the required sample size can typically be found after one test sample.

An approximation of random sampling of practical interest in microarchitecture simulation is *systematic sampling*. This approach selects sampling units from an ordered population at a fixed sampling interval k such that $n = N/k$. Systematic sampling is most effective if the population exhibits low homogeneity. In other words, the measured property χ should not vary cyclically over the population sequence at the same periodicity as k or its higher harmonics. Homogeneity in a population is quantified by the intraclass correlation coefficient δ_x; when the magnitude of δ_x is negligible, the confidence calculations for systematic sampling are the same as described for random sampling. We verified experimentally that in our sampling results the population exhibits negligible homogeneity on the order of -1×10^{-6}. This observation agrees with our intuition that realistic benchmarks do not have sufficiently regular cyclic behavior at the periodicity relevant to simulation sampling (tens of millions of instructions).

Measurement error is another source of inaccuracy for both random and systematic sampling. Random errors lead to an increase in $\hat{V}_x$ and are accounted for by a correspondingly lowered confidence in the estimate. On the other hand, systematic errors—for example, due to incorrect cache hierarchy state prior to the start of a sampling unit [11]—introduce a bias in the estimate. The bias $B(\bar{x})$ is the average difference between $\bar{X}$ and $\bar{x}$ over all possible sampling trials of a given configuration. For systematic sampling, there are exactly k possible systematic sample phases, and hence, $B(\bar{x}) = \Sigma\bar{x}/k - \bar{X}$. If bias is known, it can be accounted for by subtracting it from the estimate, without affecting confidence. If the bias can only be bounded, then it introduces a proportional amount of uncertainty in the estimate beyond the confidence interval.

3. The SMARTS framework

This section presents a framework for Sampling Microarchitecture Simulation (SMARTS). SMARTS applies statistical sampling to accelerate simulation-based performance measurements. Our presentation of SMARTS is primarily developed around estimating average CPI, but we provide results in Section 5.2 for estimating both CPI and energy. The SMARTS framework is generally applicable to other performance metrics, such as pipeline resource utilization or average memory latency.

3.1. Technique overview

Measuring the CPI of a benchmark's full instruction stream on a detailed microarchitecture simulator is a time-consuming proposition. SMARTS estimates the CPI in

1. A less rigorous, but acceptable interpretation, is that for a given sample there is a $(1-\alpha)$ probability that $\bar{x}$ is within $\pm\varepsilon \cdot \bar{X}$ of $\bar{X}$.

Table 2. SMARTS variables.

U	sampling unit size (instructions)
W	detailed warming (instructions)
N	benchmark length (instructions) / U

significantly less time by simulating and measuring only a tiny fraction of the stream on the detailed microarchitecture simulator. SMARTS assumes an execution-driven simulator that supports *detailed* simulation and *functional* simulation (a.k.a. fast-forwarding). In the detailed mode all relevant microarchitecture details are accounted for. Only programmer-visible architectural state (e.g., architectural registers and memory) is updated in the functional mode. SMARTS uses the two simulation modes to sample CPI systematically at a fixed interval—detailed simulation of the sampled instructions and functional simulation of the remaining instructions.

SMARTS uses systematic sampling rather than random sampling because systematic sampling is more straightforward to implement in execution-driven simulators. In SMARTS, a sampling unit is defined as U consecutive instructions in a benchmark's dynamic instruction stream such that the population size N is the length of the stream divided by U. The exact number of instructions per sampling unit may vary slightly to align sampling units on clock cycle boundaries. For systematic sampling at an interval k, beginning at offset j, SMARTS repeatedly alternates between a functional simulation period of $U(k-1)$ instructions and a detailed simulation/measurement period of U instructions. A primary reason we base the population on instructions rather than clock cycles is that one cannot meaningfully count the number of detailed cycles elapsed during functional simulation.

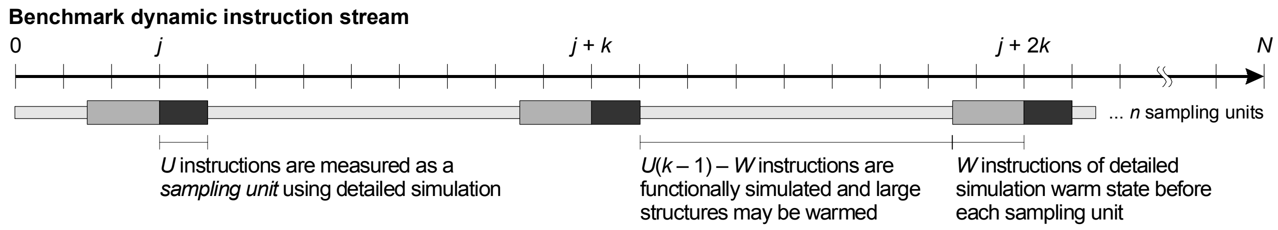
Evaluating benchmarks in SMARTS provides an estimated average CPI based on the $n \cdot U$ sampled instructions. Equally important, the results include the measured coefficient of variation $\hat{V}_{CPI}$, that allows us to calculate the confidence of the CPI estimate, and if necessary, determine a new sample size to meet a specific degree of confidence. Section 5 describes how to set SMARTS sampling parameters and prescribes an exact procedure to generate an accurate performance estimate by measuring only a minimal subset of a benchmark's instruction stream.

A key challenge in SMARTS is how to compute the correct microarchitectural state prior to detailed measurement of each sampling unit. Between sampling units, functional simulation computes all architectural state updates of the program, but leaves microarchitectural state (e.g., cache hierarchy, branch predictors and target buffers, or pipeline state) unchanged. Stale microarchitectural state introduces a large bias in the measurement of individual sampling units and, consequently, the final estimate. We have observed stale-state induced bias as high as 50% for sampling units of 10,000 instructions.

The stale-state effect can be ameliorated by introducing a warming period where W instructions are simulated in detail to refresh the microarchitectural state just prior to the measurement of a sampling unit [11]. We refer to this solution as *detailed warming*. Figure 1 graphically illustrate how SMARTS alternates between functional simulation of $[U(k-1)-W]$ instructions, detailed simulation of W warming instructions (without measurement), and detailed simulation and measurement of U instructions. Increasing W can gradually reduce the bias below an acceptable threshold.

Unfortunately, detailed warming has two major shortcomings: (1) detailed warming can be expensive because it increases the amount of detailed simulation, and (2) in general the appropriate value of W is difficult to derive analytically because some microarchitectural state has extremely long history. We will return to this discussion in Section 4.3, where we measure the effect of W on bias in a reference implementation of SMARTS.

Between detailed simulation periods, select microarchitectural state could instead be maintained by functional simulation with only a small overhead. We refer to this warming approach as *functional warming*. The cache hierarchies and branch predictors are prime candidates for functional warming. By continuously warming microarchitectural state with very long history, we can analytically bound W for the remaining state to a manageably small value. In the majority of cases, the reduction in detailed simulation more than offsets the performance overhead of functional warming. A caveat to the functional warming approach is that it may not always be able to accurately reproduce the correct microarchitectural state if correct warming requires exact knowledge of detailed execution. Moreover, timing-dependant behavior (e.g., operating



Benchmark dynamic instruction stream
0
j
j + k
j + 2k
N
... n sampling units
U instructions are measured as a sampling unit using detailed simulation
U(k – 1) – W instructions are functionally simulated and large structures may be warmed
W instructions of detailed simulation warm state before each sampling unit

Figure 1. Systematic sampling in SMARTS.

system scheduling activity) require timer approximation. If functional warming simulates instructions in order, it also may not accurately reflect the artifacts of out-of-order and speculative event ordering. A recent study [3] has suggested that out-of-order and speculative ordering has minimal impact on CPI and other performance metrics. In Section 4.5 we corroborate these results and present our own analysis of the residual biases after functional warming. We believe functional warming is the most cost-effective approach to achieve accurate CPI estimation with simulation sampling.

3.2. Benchmarks

In this paper, we demonstrate the effectiveness of SMARTS by attempting to estimate the CPI and EPI of the SPEC CPU2000 (SPEC2K) integer and floating-point benchmarks as measured on the SimpleScalar 3.0 `sim-outorder` simulator [2] with the Wattch 1.02 power estimation extensions [1]. For improved realism, we modified the memory subsystem to include a store buffer and miss status holding registers (MSHR), and model interconnect bottlenecks in the memory hierarchy. Our study includes the cross product of two microarchitecture configurations and all 26 SPEC2K benchmarks as tabulated in Figure 7. We evaluate all reference inputs except *vpr-place* and three *perlbmk* inputs, as these inputs fail to simulate correctly in `sim-outorder`. Overall, 41 benchmark/input set combinations are included in this study. To provide a reference data set for this study, we collect cycle-by-cycle traces of instruction commits in `sim-outorder` for the entire length of each benchmark. Simulating these SPEC2K benchmarks resulted in more than 7 trillion simulated instructions per machine configuration.

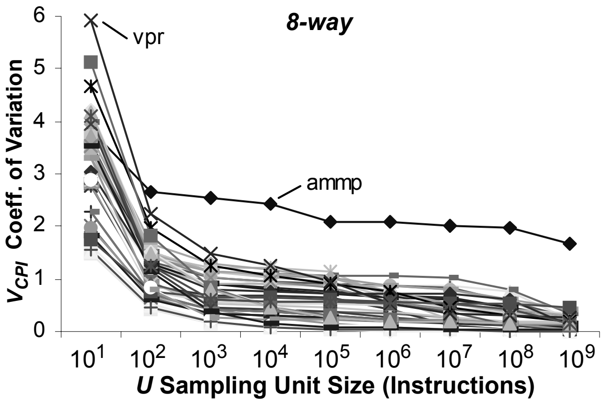


Figure 2. Coefficient of variation of CPI.

The baseline microarchitecture configuration in this study is an 8-way superscalar model that represents a processor in the current technology generation. A 16-way superscalar configuration also is included to reflect an aggressive future design point. This configuration has a wider datapath, larger out-of-order window, and larger caches, to test the effects of an enlarged state set. The details of the 8-way and 16-way configurations are summarized in Table 3.

Table 3. Machine configurations.

Parameter	8-way (baseline)	16-way
RUU/LSQ	128/64	256/128
Memory system	32KB 2-way L1I/D 2 ports, 8 MSHR 1M 4-way L2 16-entry store buffer	64KB 2-way L1I/D 4 ports, 16 MSHR 2M 8-way L2 32-entry store buffer
ITLB/ DTLB	4-way 128 entries/ 4-way 256 entries 200 cycle miss	4-way 128 entries/ 4-way 256 entries 200 cycle miss
L1/L2/mem latency	1/12/100 cycles	2/16/100 cycles
Functional units	4 I-ALU 2 I-MUL/DIV 2 FP-ALU 1 FP-MUL/DIV	16 I-ALU 8 I-MUL/DIV 8 FP-ALU 4 FP-MUL/DIV
Branch predictor	Combined 2K tables 7 cycle mispred. 1 prediction/cycle	Combined 8K tables 10 cycle mispred. 2 predictions/cycle

3.3. Speedup opportunity

The required sample size to estimate CPI at a given confidence is directly proportional to the square of the population's coefficient of variation, $n \propto V_{CPI}^2$. A benchmark with a small V_{CPI} implies a greater opportunity for accelerated simulation because fewer instructions from the benchmark need be simulated and measured in detail. To assess the potential speedup of SMARTS, we study V_{CPI} of all benchmarks in our test suite. A benchmark's instruction stream can be divided into a population using different values of U. Figure 2 plots V_{CPI} of all benchmarks on the 8-way configuration as a function of U in the range of 10 to 1 billion instructions. V_{CPI} decreases with increasing U because short-term CPI variations within a window of U instructions are hidden by averaging over the sampling unit. The V_{CPI} curves for all benchmarks share the same general shape, with a steep negative slope for U less than 1000, leveling off thereafter.

The shapes of the V_{CPI} curves argue against sampling approaches that use large sample unit sizes because for U greater than 1000, V_{CPI} (and hence n) does not decrease rapidly enough to compensate for the increased sample unit size. For instance, although very few sampling units are required in the extreme case of $U = 1 \times 10^9$, the total number of sampled instructions $n \cdot U$ is much greater than when U is less than 1000. Figure 2 further makes the case that single-sampling-unit approaches, the most commonly

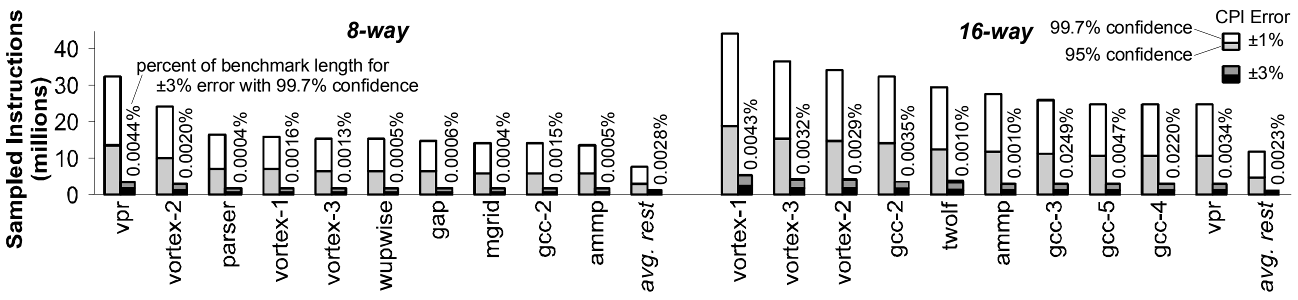


Figure 3. Minimum instructions required. This graph shows the minimum number of instructions which must be measured to achieve commonly used confidence intervals.

employed approaches, cannot ensure accurate estimates since the coefficients of variation of many benchmarks are non-negligible even for sampling units of over one billion instructions.

For $U = 10$, Figure 3 reports the values of $n \cdot U$ for all benchmarks, assuming several commonly used confidence targets. Even for a stringent confidence requirement of $\pm 1\%$ error with 99.7% confidence, the worst-case benchmark on the 8-way configuration in our study requires no more than 0.1% of its instruction stream to be measured. The number of instructions required to achieve a particular level of confidence does not vary significantly across benchmarks because, for the most part, the benchmarks have similar values of V_{CPI}. The exceedingly low detailed simulation requirement suggests that the simulation rate of SMARTS is insensitive to the speed of the detailed microarchitecture simulation. Rather, the rate depends on the speed of the functional simulation performed for the great majority of the instruction stream between sampling units. This optimistic assessment of speedup opportunity does not factor in the detailed simulation cost for microarchitectural state warming. We next present an analytical performance model for SMARTS to take into account the cost of detailed and functional warming.

3.4. Simulation speedup model

We develop a SMARTS performance model to consider the trade-off presented by functional warming. Let $S_F \equiv 1.0$ represent the simulation rate of functional simulation, and S_D represent the simulation rate of detailed simulation relative to S_F. (Therefore, $1/S_D$ is the slowdown of detailed simulation with respect to functional simulation.) The simulation rate of SMARTS using only detailed and no functional warming is given by $S_F([N - n(U + W)]/N) + S_D[(n(U + W))/N]$. This expression is a weighted average of S_F and S_D over the fraction of the instruction stream simulated functionally versus in detail. Figure 4 plots the SMARTS simulation rates for W between 0 and 10 million instructions for *gcc-1*, with $S_D = 1/60$ (corresponding to today's fastest detailed simulators) and $S_D = 1/600$ (projected simulation rate of future processor cores). The right-hand-side vertical axis estimates the corresponding runtimes on a 2 GHz Pentium 4.

The plot shows that SMARTS simulation speed decreases from S_F to S_D as W is increased; furthermore, the anticipated future S_D results in an earlier and sharper decrease. Therefore, unless W can be bounded to a reasonably small value, full benchmark measurement by simulation sampling would remain prohibitively slow.

The simulation rate of SMARTS with functional warming can be derived from the expression for detailed warming by substituting S_{FW} (the functional warming simulation rate) for S_F. Functional warming allows us to bound W to less than a few thousand instructions—sufficiently few such that detailed warming does not affect the simulation rate. This implies that the simulation rate of SMARTS with functional warming stays close to the simulation rate of S_{FW} and is relatively insensitive to the performance of the detailed simulator. In other words, the SMARTS framework enables researchers to apply otherwise prohibitively slow detailed simulators to study complete benchmarks, provided efficient functional warming is possible. In the next section, we will present our implementation of SMARTS where $S_{FW} \approx 0.55$.

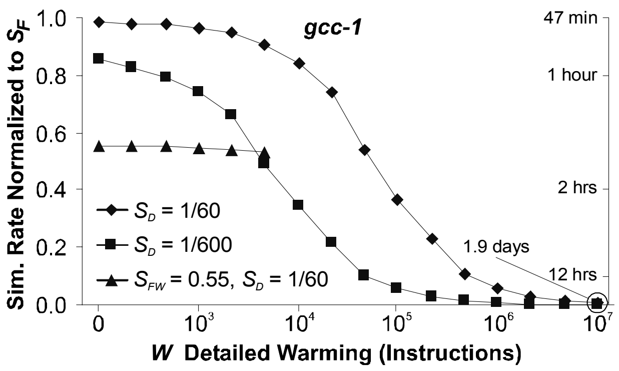


Figure 4. Modeled SMARTS simulation rate. The two S_D plots show the simulation rate without functional warming. The S_{FW} plot shows the simulation rate when using functional warming to bound W.

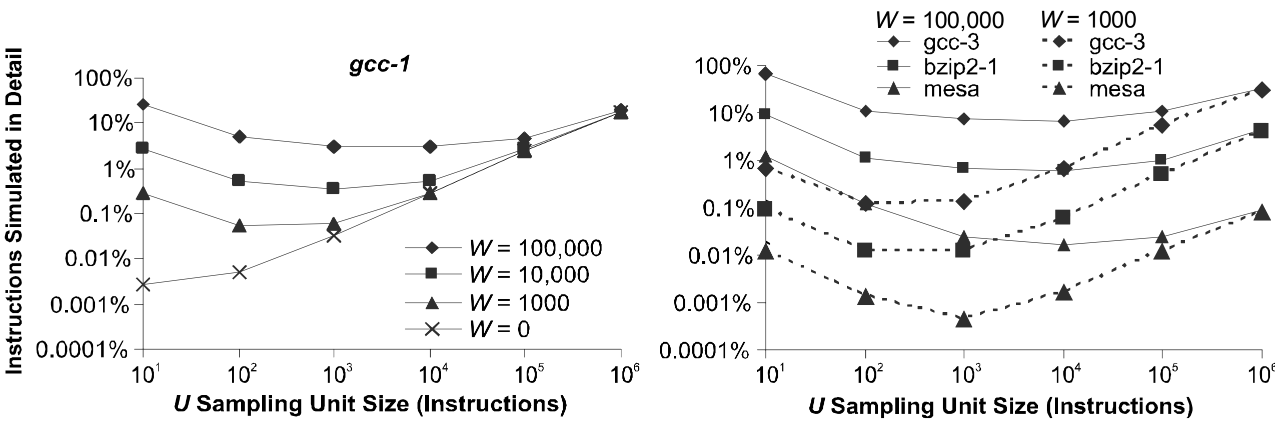


Figure 5. Optimal *U*. The left chart shows that the optimal *U* increases with *W*. The right chart shows that *U* = 1000 is a reasonable choice across benchmarks and *W*.

4. SMARTS in practice

To study and demonstrate the effectiveness of the SMARTS framework, we developed SMARTSim, a concrete implementation of a sampling microarchitecture simulator. In this section, we describe the implementation of SMARTSim and revisit the issues of microarchitectural state generation in greater detail. In particular, we explain the effect of detailed warming on the choice of sampling unit size and analyze the effectiveness of detailed warming and functional warming in generating accurate microarchitectural state for sample measurements.

4.1. SMARTSim

SMARTSim is built on our enhanced `sim-outorder` as described in Section 3.2. `Sim-outorder` supports a functional simulation mode, similar to the operation of `sim-fast` in SimpleScalar, that runs approximately 60 times faster than detailed simulation. However, `sim-outorder` only supports functional simulation prior to starting detailed simulation. SMARTSim allows repeated transitions back-and-forth between functional and detailed simulation modes.

SMARTSim accepts `sim-outorder` command line arguments and configuration files. In addition, SMARTSim accepts the systematic sampling parameters *U*, *k*, *W*, and *j* (described in Section 3.1). SMARTSim also supports two fast-forwarding options: functional simulation only and functional simulation with warming (a.k.a. functional warming). For functional warming, SMARTSim performs in-order functional instruction execution and maintains the state of L1/L2 I/D caches, TLBs, and branch predictors in a fashion similar to `sim-cache` and `sim-bpred` of SimpleScalar. In SMARTSim, functional warming operations introduce an overhead of approximately 75% over functional simulation alone.

4.2. Optimal sampling unit size

SMARTSim allows the user to specify the sampling unit size *U*. In the analysis in Section 3.3, we have shown that smaller unit sizes reduce the number of instructions simulated in detail if the cost of detailed warming is ignored. However, because detailed warming adds an overhead of *W* instructions of detailed simulation per sampling unit, the optimal value for *U* increases with increased *W* to amortize the overhead of detailed warming. To illustrate the effect of *W* on the choice of *U*, Figure 5 (left) plots the fraction of instructions simulated in detail (i.e., $n(W + U)/N$) for various values of *U* and *W*. The data points are based on SMARTSim execution of *gcc-1* on the 8-way configuration, with *n* chosen for 99.7% confidence interval of ±3% in the CPI estimate. In the idealized case where $W = 0$, the minimum *U* leads to the fewest detail-simulated instructions. For non-ideal *W*, however, the optimal value of *U* lies in the range of 100 to 10,000 instructions. Figure 5 (right) locates the optimal values of *U* for three other benchmarks, *gcc-3*, *bzip2-1*, and *mesa.* Each benchmark is plotted for two values of *W* (1000 and 100,000) that are approximately the magnitudes needed for sampling with and without functional warming, as discussed in the following two sections. The optimal choice of *U* is not fixed across benchmarks. However, in all cases, including other SPEC2K benchmarks not shown, fixing *U* to 1000 leads to a sufficiently small fraction of detail-simulated instructions such that choosing the optimal *U* gains at most tens of minutes in SMARTSim run time. Therefore, we suggest using $U = 1000$ in all cases.

4.3. Effectiveness of detailed warming

Microarchitectural state can always be warmed to an arbitrary degree of accuracy given sufficient detailed warming. Unfortunately, the required amount of detailed warming to obtain a given degree of accuracy cannot be determined analytically. The required amount is a function

Table 4. Detailed warming requirements without functional warming. (8-way)

***W* to achieve < 1.5% bias**	**Benchmarks**
$W \leq 50 \times 10^3$	applu, apsi, art-1, art-2, eon-1, eon-2, equake, fma3d, gzip-1, gzip-2, gzip-3, gzip-4, lucas, mesa, sixtrack, twolf
$W \leq 250 \times 10^3$	crafty, eon-3, gap, gcc-1, gcc-3, gcc-4, mcf, swim, vortex-3, vpr
$W \leq 500 \times 10^3$	ammp, bzip2-1, bzip2-2, galgel, gcc-2, gcc-5, gzip-5, vortex-1, vortex-2
$W > 500 \times 10^3$	bzip2-3, facerec, mgrid, parser, perlbmk, wupwise

of both the benchmark behavior and the microarchitectural mechanisms involved. As a rule of thumb, we expect the amount of detailed warming to scale with the size of the microarchitectural state; however, there are counter-examples.

To better understand the requirements of detailed warming (unaided by functional warming), we experimentally determine the minimum acceptable value of *W* for the benchmarks with the 8-way configuration such that the bias due to residual microarchitectural state error is just below ±1.5%. (We choose $U = 1000$ and *n* sufficient for a 99.7% confidence interval of ±3%.) In systematic sampling, the true bias is the average error over all *k* possible systematic samples. Exact determination of bias is prohibitively expensive, since *k* is typically on the order of 10,000 in this study. Therefore, we approximate the procedure by averaging the errors of 5 evenly distributed systematic sampling runs (i.e., $j = \{0, k/5, 2k/5, 3k/5, 4k/5\}$). Table 4 categorizes the studied benchmarks according to their required values of *W*.

Without functional warming, the required *W* varies widely across benchmarks and inputs. Many benchmarks are insensitive to the accuracy of microarchitectural state, requiring less than 50,000 instructions of detailed warming per measurement period. For some benchmarks, however, even $W = 500{,}000$ results in unacceptable bias, as high as 25% for mgrid.

With the exception of the benchmarks requiring more than 500,000 instructions of detailed warming, detailed warming does not significantly impact the simulation rate of SMARTSim. Even 500,000 instructions warmed per sampling unit is a small fraction of the full benchmark. Nevertheless, Table 4 does highlight a key shortcoming of the detailed-warming-only approach: the unpredictability of *W*. Our empirical determination of *W* is impractical because it requires *a priori* knowledge of the true unbiased CPI derived from prohibitively time-consuming detailed simulation of complete benchmarks.

4.4. Bounding detailed warming

Functional warming helps redress the unpredictability of *W* in detailed warming. Functional warming of problematic microarchitectural state allows us to bound *W* safely for the remaining state by analyzing the details of the microarchitecture model. For example, to estimate CPI, *W* needs to be chosen such that an instruction's latency cannot be influenced by unwarmed microarchitectural state. This requires *W* to exceed the maximum instruction stream distance that latency-influencing state can propagate.

An instruction can only affect the latency of another instruction if there is some history of the former still present at the time the latter is fetched. Outside of long-term architectural (register, memory, etc.) and microarchitectural state (cache, TLB, branch predictor, etc.) maintained by functional warming, the effects of an instruction are bounded by the instruction's lifetime in the microprocessor. With the exception of store instructions, when an instruction commits, its associated short-term state is freed. A committed store instruction that misses in the cache might stall a later store instruction by causing the store buffer to overflow. Hence, a worst-case bound on *W* is the product of store-buffer depth, memory latency in cycles, and the maximum IPC. For our 8-way configuration, this upper bound is 12,800 ($16 \times 100 \times 8$) instructions. In practice, this worst-case behavior does not occur; all the 8-way results presented in this paper were achieved with only 2000 instructions of detailed warming, and 16-way results with 4000.

4.5. Effectiveness of functional warming

Even with both functional and detailed warming, some inaccuracies in microarchitectural state remain and contribute to errors in the estimates as bias. Table 5 reports the residual bias in the CPI estimated by SMARTSim when functional warming is employed in conjunction with

Table 5. CPI bias achieved with functional warming and minimal detailed warming.

8-way	vpr	galgel	gcc-2	bzip2-2	parser	gzip-5	facerec	gcc-5	vortex-3	gcc-1	*avg. rest (abs)*
***W* = 2000**	-1.6%	1.4%	-1.1%	-1.0%	1.0%	0.9%	0.9%	-0.8%	-0.6%	-0.5%	0.2%
16-way	mcf	gcc-2	vortex-3	eon-2	gcc-5	sixtrack	wupwise	bzip2-1	applu	mesa	*avg. rest (abs)*
***W* = 4000**	1.9%	-1.6%	1.2%	-1.1%	-1.1%	-0.9%	0.9%	0.8%	0.7%	-0.6%	0.2%

detailed warming of the aforementioned values of W. Benchmarks with the worst bias are presented in sorted-order. The final column of the table gives the average magnitude of remaining benchmarks' bias. All benchmarks have bias under ±2.0% and only 6 benchmarks in each configuration exceed ±1.0%. The bias is predominantly due to wrong-path and out-of-order effects in caches and the branch predictor. This set of results corroborates our conclusion that functional-warming with bounded W is effective in reducing microarchitectural state warming bias.

5. Using SMARTS

This section outlines an exact procedure for estimating a target metric using statistical simulation sampling. We evaluate the effectiveness of this procedure by estimating the CPI and energy per instruction (EPI) of SPEC2K using SMARTSim.

5.1. SMARTS procedure

One iteration of a SMARTS measurement run requires the user to supply three sampling simulation parameters: W, U, and k. First, W is selected to exceed the bounded history of the microarchitectural state as described in Section 4.4. We recommend utilizing functional warming (see Section 4.5) whenever possible, as it greatly simplifies the determination of W. Our 8-way results were achieved with $W = 2000$ instructions, and 16-way results with $W = 4000$. Second, we suggest setting $U = 1000$. We have shown in Section 4.2 that $U = 1000$ is appropriate for all SPEC2K benchmarks. Lastly, we elaborate on how to determine n, and correspondingly k, to meet a desired confidence in the following paragraphs.

In general, the correct value for n must be determined in a two-step process. First, a sampling measurement is made using a generic initial value n_{init} that is a compromise between simulation rate and the likelihood of meeting the confidence requirement on the first try. If the choice of n_{init} is shown to be insufficient after one sampling simulation, a second step is required where n_{tuned} for a second run is calculated from the $\hat{V}_x$ of the initial run.

A priori, the minimum value of n to achieve a given confidence is unknown for an arbitrary benchmark and simulated microarchitecture. Given a fixed confidence target, n must be adjusted according to the coefficient of variation V_{CPI} of the population. Based on our analysis of V_{CPI} of SPEC2K benchmarks (in Section 3.3), we conjecture that the values of V_{CPI} tend to cluster around 1.0 for most benchmarks and simulated microarchitectures when $U = 1000$. Hence, from $n_{init} = (z/\varepsilon)^2$, we infer that $n_{init} = 10{,}000$ is likely to yield 99.7% confidence interval of ±3%. Given $N = 9{,}420{,}910$ for the smallest of our SPEC2K benchmarks, $n_{init} = 10{,}000$ still represents a very small fraction of detail-simulated instructions and hence has minimal impact on simulation turnaround time.

One run of SMARTS measurement with $k = N/n_{init}$ produces an initial estimate of average CPI and $\hat{V}_{CPI}$ of the sample. Because the confidence of an estimate is jointly quantified by the two interdependent terms confidence level $(1-\alpha)$ and confidence interval $\pm\varepsilon \cdot \overline{X}$, one can either set a desired confidence level and calculate the obtained confidence interval for a given sample, or vice versa. For a set confidence level $(1-\alpha)$, the confidence interval is $\pm(z \cdot \hat{V}_x \cdot \bar{x})/(\sqrt{n})$ where z is the $100[1-(\alpha/2)]$ percentile of the standard normal distribution. Commonly used confidence levels are 95% and 99.7% (a.k.a. 3σ or *virtually-certain*). Corresponding values of z are 1.97 and 3, respectively. If the confidence level and interval yielded by the initial sample are unacceptable, the n_{tuned} to achieve a desired confidence on the next sample is $((z \cdot \hat{V}_x)/\varepsilon)^2$. If the initial confidence is overly below target, we suggest slightly overestimating n_{tuned} for the subsequent run. In any case, the actual confidence achieved by the subsequent sample must be checked using the subsequent sample's new $\hat{V}_{CPI}$.

The above treatment of confidence considers only the error introduced by statistical sampling. In practice, the true error margin in an estimate must also account for any bias in the measurements. Recall from Section 2 that if the bias is known, it can be accounted for by subtracting it

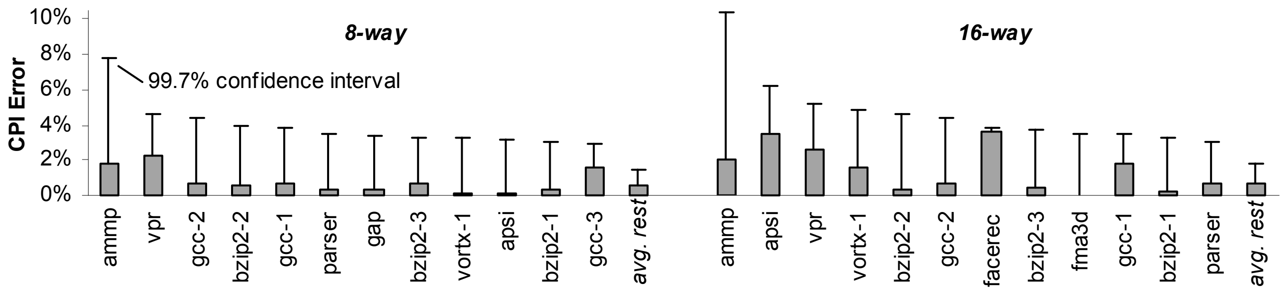


Figure 6. SMARTS results across SPEC2K with *n* = 10,000. Unacceptably large confidence intervals (e.g., 8-way ammp, vpr, and gcc-2) can be improved by simulating with n_{tuned}.

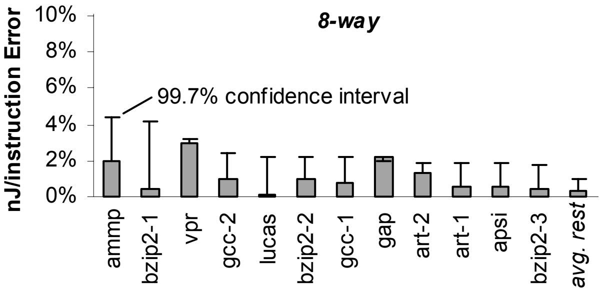


Figure 7. SMARTS EPI results with *n* = 10,000.

from the estimate, without affecting confidence. If the bias can only be bounded, then it introduces a proportional amount of uncertainty in the estimate beyond the confidence interval.

5.2. Evaluation of performance and accuracy

We applied the procedure outlined above to SPEC2K benchmarks using SMARTSim. Figure 6 reports results of CPI estimated using SMARTSim in one run with n_{init} = 10,000. Benchmarks with the worst confidence intervals are shown in sorted order, plus the average of the remaining benchmarks. For each benchmark, we show the actual achieved error and the predicted confidence interval calculated from $\hat{V}_x$ for 99.7% confidence. The confidence interval accounts for random error in the estimated CPI that is introduced by systematic sampling. Notice that actual error resulting from 10,000 sampling units is generally much less than the predicted confidence interval. A large part of this error can be attributed to the residual bias of imperfect microarchitectural state warming (functional warming with fixed *W)*, with only a very small component caused by statistical sampling.

For most of the benchmarks, n_{init} achieves a confidence interval within ±3%. For benchmarks with confidence intervals greater than ±3%, simulation sampling needs to be repeated using n_{tuned}—calculated from the $\hat{V}_x$ of the initial sample. For example, rerunning simulations for the 8-way configuration with n_{tuned} of 66,531 (*ammp*), 23,321 (*vpr*), and 21,789 (*gcc-2*) achieve actual errors of 1.1%, 0.1%, and -0.9% with confidence intervals of 3.0%, 2.9%, and 2.6%. To this confidence interval, we must still add an uncertainty due to microarchitectural state warming bias, which we empirically bound to below 2%.

Figure 7 presents the results of applying SMARTS to estimating energy per instruction (EPI). As in CPI estimations, we find in most cases initial sampling simulations using n_{init} = 10,000 achieves confidence intervals tighter than ±3%. Confidence intervals for EPI estimation tend to be tighter than CPI confidence intervals because of less variability in EPI. Unfortunately, the smaller predicted confidence intervals are overshadowed by the microarchitectural state warming bias. With the exception of *gap*, the actual errors are within the confidence interval. For *gap*, we have determined experimentally that the 2.2% error is almost entirely due to bias.

Table 6 compares simulation runtimes for functional (i.e., `sim-fast`), detailed (i.e., `sim-outorder` with detailed memory models), and SMARTSim simulation on a 2 GHz Pentium 4. SPEC2K benchmarks on the 8-way configuration with the highest instruction counts are shown in sorted order. As shown in Table 6, detailed simulation takes on average 7.2 days and can take as long as 23 days. In contrast, SMARTSim takes on average 5.0 hours and in the worst-case slightly less than 16 hours. SMARTSim simulation speed is around 50% of functional-only simulation for most microarchitecture configurations.

5.3. Comparison to SimPoint

A recent proposal, SimPoint [17], also enables reduced simulation turnaround time. SimPoint selects representative subsets of benchmark traces via offline analysis of basic blocks. Using clustering algorithms, SimPoint selects and weights several large sampling units (up to ten 100M-instruction sampling units) such that the frequency of each static basic block across the weighted units matches that block's frequency in the full dynamic stream. A fundamental assumption of SimPoint is that all dynamic instances of basic block sequences with similar profiles have the same behavior, therefore a particular sequence can be measured once and weighted appropriately to represent all remaining instances.

SimPoint has two key advantages: (1) due to large sampling units, SimPoint obviates the need for functional warming and can be more quickly integrated into a simulation infrastructure, and (2) SimPoint allows early termination of simulation after all selected sections have been visited. We implemented SimPoint with our SimpleScalar toolset and verified our implementation against the published configuration and results in [17]. For

Table 6. Runtimes for SMARTS compared to detailed and functional simulation. (8-way)

Runtime (hrs.)	parser	sixtrack	mgrid	galgel	wupwise	apsi	twolf	ammp	mesa	gap	fma3d	swim	*avg. rest*
Detailed	541	466	414	405	346	344	343	323	279	266	265	223	98
Functional	9.2	7.9	7.0	6.9	5.9	5.8	5.8	5.5	4.7	4.5	4.5	3.8	1.7
SMARTS	15.8	13.6	12.1	11.8	10.1	10.1	10.0	9.6	8.1	7.8	7.8	6.5	2.9

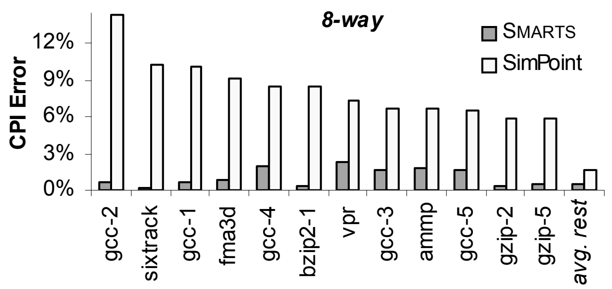


Figure 8. Comparison of SMARTS with SimPoint.
SimPoint's mean runtime per benchmark is 2.8 hours compared to 5.0 hours for SMARTS.

the benchmarks in [17] and our 8-way configuration, SimPoint resulted in an average improvement of 1.8 in simulation rate over SMARTS.

However, SimPoint has several shortcomings: (1) it may result in arbitrarily high CPI error, (2) it does not offer quantifiable confidence in estimates, and (3) some microarchitecture configurations may cause large variations in behavior across different instances of similarly-profiled basic block sequences.[1]

Figure 8 presents a comparison of CPI error between SimPoint and SMARTS for the benchmarks presented in [17] running on our 8-way configuration. The comparison shows that SimPoint has a higher average error (3.7% vs. our 0.6%) and considerably higher worst-case error (-14.3% for *gcc-2*).

Gcc-2 is an example where SimPoint produces an unacceptably high CPI error when running on our 8-way configuration. However, simulation using the published microarchitecture configuration in [17] only results in a 1.6% error. In *gcc-2*, we observed that the basic block sequences chosen by SimPoint exhibit large variations in their L2 miss rate—due to variations in data cache locality—across dynamic instances on our microarchitecture configuration. Therefore, in this case, the SimPoint estimate based on just a single instance of the basic block sequences yields a large error. In contrast, independent of benchmark and microarchitecture configuration, SMARTS uses the measured coefficient of variation to help gauge both the required sample size and the confidence in the estimates.

1. Consider a basic block comprised of a pointer-chasing loop. The execution time of each dynamic instance depends on whether the pointer deference hits in the cache and hence is a function of the cache design and the precise memory placement of the linked list nodes.

6. Conclusion

To address the need for improved simulation accuracy and performance, we propose the Sampling Microarchitecture Simulation (SMARTS) framework that applies statistical sampling to microarchitecture simulation. Unlike prior approaches to simulation sampling, SMARTS prescribes an exact and constructive procedure for sampling a minimal subset of a benchmark's instruction execution stream to estimate the performance of the complete benchmark with quantifiable confidence. The SMARTS procedure obviates the need for full-stream simulation by basing the strategy for optimal simulation sampling on the outcomes of fast sampling simulation runs.

We evaluated the SMARTS framework in the context of a wide-issue out-of-order superscalar simulator running SPEC2K benchmarks with varying inputs under two simulated processor configurations. SMARTSim, an implementation of SMARTS, is created by modifying SimpleScalar's `sim-outorder` to support systematic sampling. The results of our evaluations demonstrated the following: (1) SMARTSim achieves an actual average error of only 0.64% on CPI and 0.59% on EPI by simulating fewer than 50 million instructions in detail per benchmark. (2) By simulating exceedingly small fractions of complete benchmarks, SMARTSim achieves effective speeds of 9.2 MIPS and 9.0 MIPS simulating 8-way and 16-way out-of-order processors on a 2 GHz Pentium 4. This corresponds to speedups of 35 and 60 times over full-stream simulation with `sim-outorder` for the two configurations.

The outcomes of this study have two fundamental bearings on future simulator designs. First, designers should not attempt to accelerate detailed simulators at the cost of coding complexity or abstraction errors; instead designers should focus on increasing the simulator's flexibility and realism. Second, designers should focus on techniques to speed up fast-forwarding and functional warming, because these ultimately determine sampling simulation time.

Acknowledgment

The authors would like to thank Se-Hyun Yang, Zeba Wunderlich, the members of the Carnegie Mellon Impetus group, and the anonymous reviewers for their feedback on earlier drafts of this paper. This work was funded in part by grants from IBM and Intel corporations, an NSF CAREER award, and an NSF Instrumentation award.

References

[1] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," In *Proceedings of the International Symposium on Computer Architecture*, June 2000.

[2] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report 1342, Computer Sciences Department, University of Wisconsin–Madison, June 1997.

[3] H. W. Cain, K. M. Lepak, B. A. Schwartz, and M. H. Lipasti, "Precise and Accurate Processor Simulation," In *Workshop on Computer Architecture Evaluation using Commercial Workloads, HPCA*, February 2002.

[4] T. M. Conte, M. A. Hirsch, and K. N. Menezes, "Reducing State Loss for Effective Trace Sampling of Superscalar Processors," In *Proceedings of the International Conference on Computer Design*, October 1996.

[5] M. Durbhakula, V. S. Pai, and S. Adve, "Improving the Accuracy vs. Speed Tradeoff for Simulating Shared-Memory Multiprocessors with ILP Processors," In *Proceedings of the International Symposium on High-Performance Computer Architecture*, January 1999.

[6] S. Dwarkadas, J. R. Jump, and J. B. Sinclair, "Execution-Driven Simulation of Multiprocessors: Address and Timing Analysis," *IEEE Transactions on Modeling and Computer Simulation*, Volume 4, No. 4, October 1994.

[7] J. W. Haskins and K. Skadron, "Minimal Subset Evaluation: Rapid Warm-Up for Simulated Hardware State," In *Proceedings of the International Conference on Computer Design*, September 2001.

[8] W. C. Hsu, H. Chen, and P. C. Yew, "On the Predictability of Program Behavior Using Different Input Data Sets," In *Workshop on Interaction between Compilers and Computer Architectures, HPCA*, February 2002.

[9] AJ KleinOsowski, J. Flynn, N. Meares, and D. J. Lilja, "Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research," In *IEEE Workshop on Workload Characterization, ICCD*, September 2000.

[10] T. Lafage and A. Seznec, "Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Stream," In *IEEE Workshop on Workload Characterization, ICCD*, September 2000.

[11] S. Laha, J. H. Patel, and R. K. Iyer, "Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems," *IEEE Transactions on Computers*, Volume C-37(11), February 1988.

[12] G. Lauterbach, "Accelerating Architectural Simulation by Parallel Execution of Trace Samples," In *Hawaii International Conference on System Sciences*, Volume 1: Architecture, January 1994.

[13] P. S. Levy and S. Lemeshow, *Sampling of Populations: Methods and Applications*, John Wiley & Sons, Inc., 1999.

[14] S. Nussbaum and J. E. Smith, "Modeling Superscalar Processors via Statistical Simulation," In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

[15] M. Oskin, F. T. Chong, and M. K. Farrens, "HLS: Combining Statistical and Symbolic Simulation to Guide Microprocessor Designs," In *Proceedings of the International Symposium on Computer Architecture*, June 2000.

[16] S. K. Reinhardt, M. D. Hill, J. R. Larus, A. R. Lebeck, J. C. Lewis, and D. A. Wood, "The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers," In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, May 1993.

[17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, October 2002.

[18] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, J. C. Hoe. "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," In *Proceedings of the International Symposium on Computer Architecture*, June 2003.

Appendix A: Additional Results

Table 7. SPEC CPU2000 Benchmarks

Benchmark	Input	Instructions (bil.)	8-way IPC	16-way IPC	8-way EPI (nJ/Inst.)
ammp		326.5	1.04	1.60	42.7
applu		223.9	1.17	1.75	42.1
apsi		347.9	1.58	2.40	35.9
art-1	startx 110	41.8	0.39	0.66	88.0
art-2	startx 470	45.0	0.39	0.66	87.6
bzip2-1	source	108.9	1.48	1.77	39.5
bzip2-2	graphic	143.6	1.56	1.95	37.2
bzip2-3	program	124.9	1.70	2.08	36.9
crafty		191.9	2.34	2.94	34.6
eon-1	kajiya	101.3	2.50	3.11	36.5
eon-2	cook	80.6	3.01	4.81	31.8
eon-3	rushmeier	57.9	2.85	4.11	33.2
equake		131.5	0.79	1.23	50.3
facerec		211.0	1.86	3.31	33.2
fma3d		268.4	1.53	2.57	38.0
galgel		409.4	0.96	1.77	45.2
gap		269.0	1.48	1.74	39.5
gcc-1	166	46.9	1.45	1.41	40.7
gcc-2	200	108.6	1.60	1.82	39.6
gcc-3	expr	12.1	1.63	1.73	40.1
gcc-4	integrate	13.2	1.64	1.62	38.9
gcc-5	scilab	62.0	1.64	1.80	40.2
gzip-1	source	84.4	1.76	1.91	37.2
gzip-2	log	39.5	1.81	1.94	35.9
gzip-3	graphic	103.7	2.26	2.54	35.7
gzip-4	random	82.2	2.22	2.48	36.0
gzip-5	program	168.9	1.81	2.00	36.1
lucas		142.4	0.11	0.11	207.1
mcf		61.9	0.10	0.14	245.2
mesa		281.7	2.92	4.44	29.6
mgrid		419.2	1.75	2.91	36.3
parser		546.7	1.32	1.58	41.4
perlbmk	makerand	2.1	2.01	2.27	36.4
sixtrack		470.9	2.50	5.79	31.1
swim		225.8	1.03	1.51	42.8
twolf		346.5	0.76	0.83	52.7
vortex-1	lendian1	119.0	2.13	3.38	31.4
vortex-2	lendian2	138.7	2.35	3.89	30.7
vortex-3	lendian3	133.0	2.12	3.36	31.5
vpr	route	84.1	0.56	0.64	63.8
wupwise		349.6	2.37	4.26	30.4

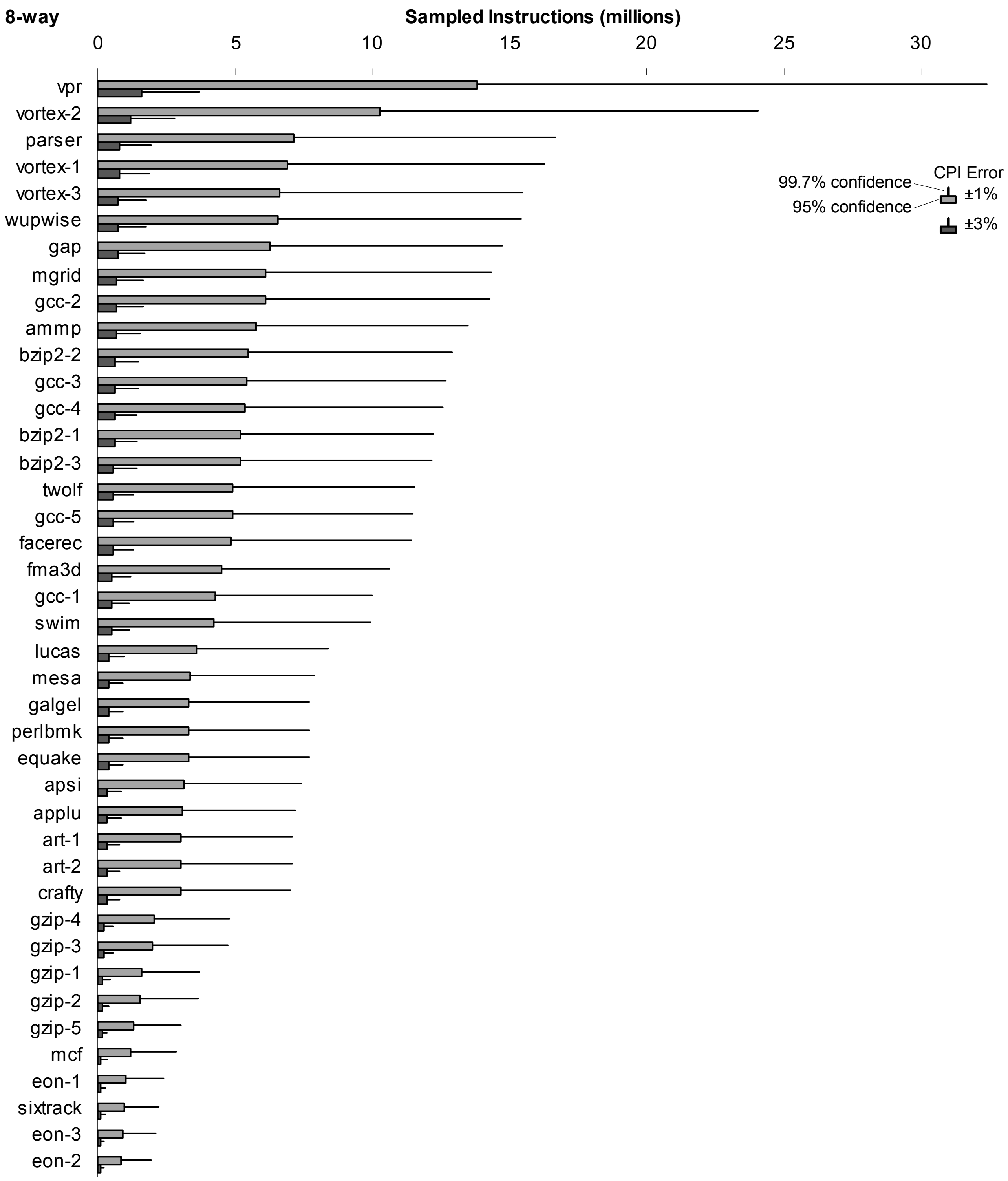


Figure 9. Minimum instructions required.

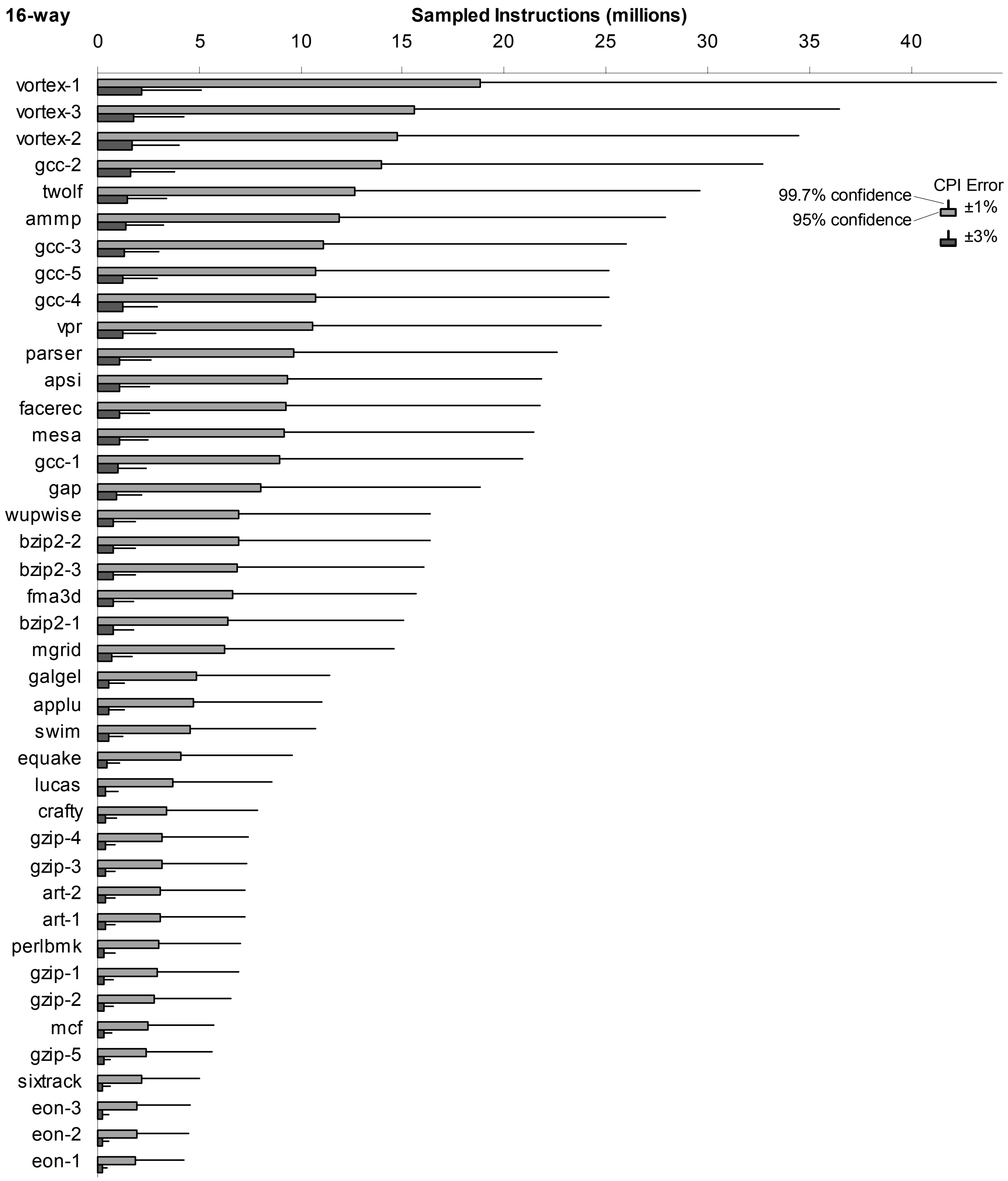
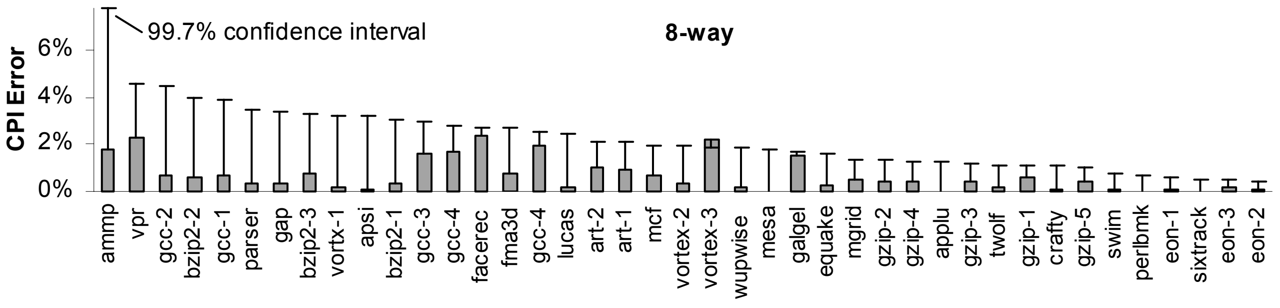


Figure 10. Minimum instructions required.

Table 8. Bias achieved with functional warming and minimal detailed warming.

8-way	CPI Bias	EPI Bias	8-way	CPI Bias	EPI Bias	16-way	CPI Bias	16-way	CPI Bias
vpr	-1.56%	0.52%	vortex-1	-0.29%	0.80%	mcf	1.88%	gzip-1	-0.25%
galgel	1.37%	0.04%	gcc-4	-0.29%	-0.09%	gcc-2	-1.60%	galgel	-0.25%
gcc-2	-1.07%	0.63%	mgrid	0.28%	0.09%	vortex-3	1.18%	gcc-4	0.24%
bzip2-2	-1.04%	0.94%	bzip2-1	-0.25%	0.55%	eon-2	-1.11%	bzip2-2	-0.19%
parser	1.01%	0.56%	gzip-3	-0.25%	0.05%	gcc-5	-1.10%	mgrid	-0.17%
gzip-5	0.94%	2.31%	ammp	0.18%	-2.42%	sixtrack	-0.93%	art-1	-0.15%
facerec	0.86%	0.96%	sixtrack	0.17%	-0.27%	wupwise	0.85%	gzip-2	0.14%
gcc-5	-0.81%	0.04%	wupwise	-0.17%	-0.05%	bzip2-1	0.78%	gzip-5	-0.12%
vortex-3	-0.55%	0.63%	equake	0.13%	1.46%	applu	0.65%	vortex-2	-0.12%
gcc-1	-0.53%	-1.03%	applu	-0.12%	-0.04%	mesa	-0.58%	lucas	0.09%
bzip2-3	-0.51%	0.36%	gzip-4	-0.11%	0.09%	eon-1	-0.56%	art-2	0.07%
perlbmk	-0.40%	-0.09%	eon-2	-0.10%	-0.10%	vortex-1	-0.54%	apsi	-0.07%
swim	0.38%	0.19%	twolf	0.09%	0.00%	ammp	-0.53%	parser	0.06%
gzip-1	0.38%	1.43%	gzip-2	-0.09%	0.23%	swim	0.44%	gzip-3	-0.05%
mcf	0.36%	1.14%	mesa	-0.07%	0.11%	vpr	0.38%	twolf	0.04%
eon-1	-0.36%	-0.14%	gap	0.07%	2.49%	gcc-3	-0.36%	bzip2-3	-0.04%
fma3d	-0.35%	-0.22%	gcc-3	-0.05%	0.00%	crafty	0.32%	eon-3	0.04%
crafty	-0.35%	-0.14%	eon-3	-0.04%	-0.15%	perlbmk	-0.30%	facerec	0.03%
art-2	0.31%	0.13%	lucas	0.03%	0.13%	fma3d	0.28%	equake	0.02%
art-1	-0.30%	-0.40%	vortex-2	0.02%	1.09%	gap	0.28%	gzip-4	0.00%
apsi	0.29%	-0.42%				gcc-1	0.25%		



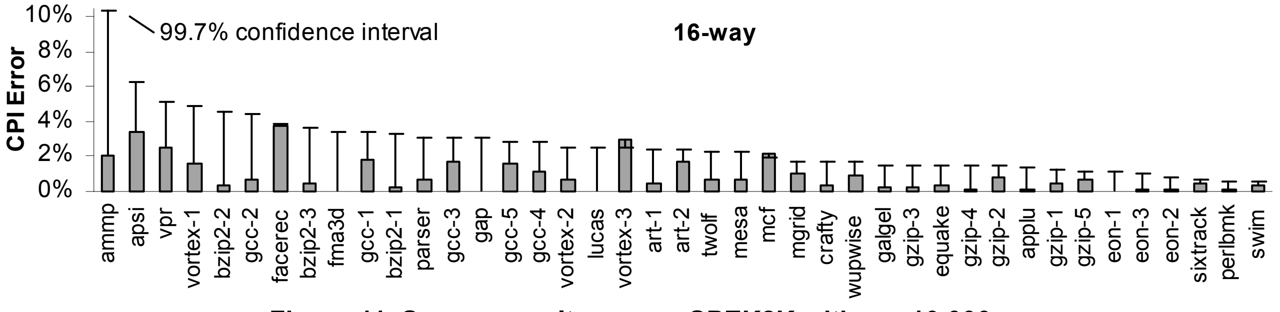


Figure 11. SMARTS results across SPEK2K with n = 10,000.

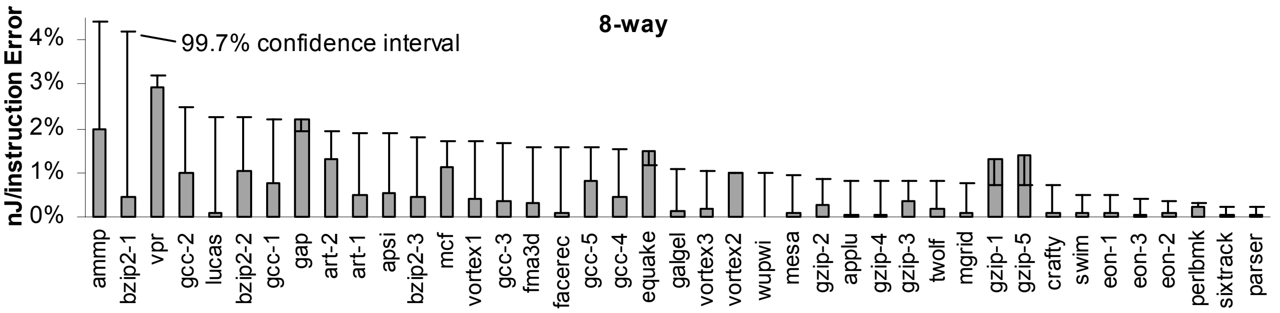


Figure 12. SMARTS EPI results with n = 10,000.

Table 9. Runtimes for SMARTS compared to detailed and functional simulation. (8-way)

Runtime (hrs.)	Detailed	Functional	SMARTS	Runtime (hrs.)	Detailed	Functional	SMARTS
parser	541	9.2	15.8	bzip2-3	123	2.1	3.6
sixtrack	466	7.9	13.6	vortex-1	118	2.0	3.5
mgrid	414	7.0	12.1	bzip2-1	108	1.8	3.2
galgel	405	6.9	11.8	gcc-2	107	1.8	3.2
wupwise	346	5.9	10.1	gzip-3	103	1.7	3.0
apsi	344	5.8	10.1	eon-1	100	1.7	2.9
twolf	343	5.8	10.0	gzip-1	83	1.4	2.4
ammp	323	5.5	9.6	vpr	83	1.4	2.5
mesa	278	4.7	8.1	gzip-4	81	1.4	2.4
gap	266	4.5	7.8	eon-2	80	1.4	2.3
fma3d	265	4.5	7.8	gcc-5	61	1.0	1.8
swim	223	3.8	6.5	mcf	61	1.0	1.8
applu	221	3.8	6.5	eon-3	57	1.0	1.7
facerec	209	3.5	6.1	gcc-1	46	0.8	1.4
crafty	190	3.2	5.5	art-2	45	0.8	1.3
gzip-5	167	2.8	4.9	art-1	41	0.7	1.2
bzip2-2	142	2.4	4.2	gzip-2	39	0.7	1.2
lucas	141	2.4	4.1	gcc-4	13	0.2	0.4
vortex-2	137	2.3	4.0	gcc-3	12	0.2	0.4
vortex-3	132	2.2	3.9	perlbmk	2	0.0	0.1
equake	130	2.2	3.8				

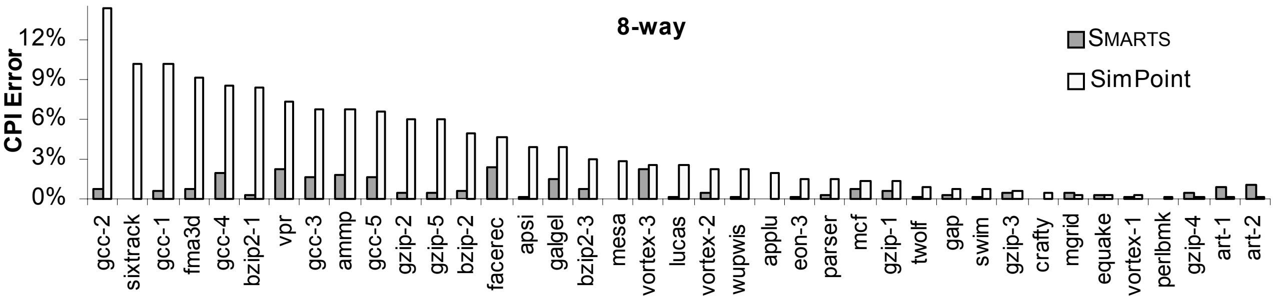


Figure 13. Comparison of SMARTS with SimPoint.

Table 10. Statistical analysis of SimPoint clusters for gcc-2. (8-way as in [17])

Cluster	Weight	Actual IPC	IPC of SimPoint	V_{CPI}
Full benchmark	—	0.96	—	0.49
1	12.155%	2.93	3.69	0.52
2	18.785%	1.09	0.97	0.33
3	11.602%	0.64	0.52	0.18
4	47.053%	0.77	0.78	0.28
5	5.893%	2.58	2.98	0.56
6	4.512%	2.60	2.43	0.38

Appendix B: Statistical Analysis of SimPoint

In this section, we present a statistical analysis of the SimPoint clustering and selection technique presented in [17]. SimPoint attempts to select less than 10 sampling units, with unit size of 100M instructions each, to represent a benchmark. To select these sampling units, a benchmark's complete dynamic trace is divided into units of 100M instructions. The units are grouped into up to 10 clusters based on the similarity of their basic block vectors (defined in [17]). From each cluster, SimPoint selects a single unit to represent that cluster. The key premise behind SimPoint is that similarity in basic block vectors correlate strongly to similarity in IPC. Therefore, a weighted average IPC of only the selected units can estimate a benchmark's overall IPC.

We measure V_{CPI} across the units assigned to each cluster to evaluate statistically how well full-benchmark system performance is represented. Because units in each cluster have similar basic block vectors, we expect the V_{CPI} within a single cluster to be low. Table 10 shows V_{CPI} for each of the six clusters for *gcc-2.* The data in the table is derived from simulations using the microarchitecture configuration in [17]. We reproduced the cluster assignments of [17] using the provided SimPoint analysis tools. In the case of clusters 1 and 5, the coefficient of variation within the cluster exceeds the coefficient of variation for the whole benchmark. Therefore, the probability that any single unit selected from the cluster accurately represents the cluster is low. For both of these clusters, the difference between the average IPC over all of the cluster's units and the IPC of the selected unit is large (26% and 16% respectively).

Figure 14 depicts the sampling units assigned to the two highest-weighted clusters in *gcc-2.* The figure illustrates the large amount of variation within each cluster. The single unit chosen to represent each cluster is marked.

These results also indicate that the clustering technique provides a rather low confidence in the estimates. We find that achieving 95% confidence of ±3% CPI error requires on the order of 1,000 sampling units (given a V_{CPI} of approximately 0.5 for a unit size of 100M as seen

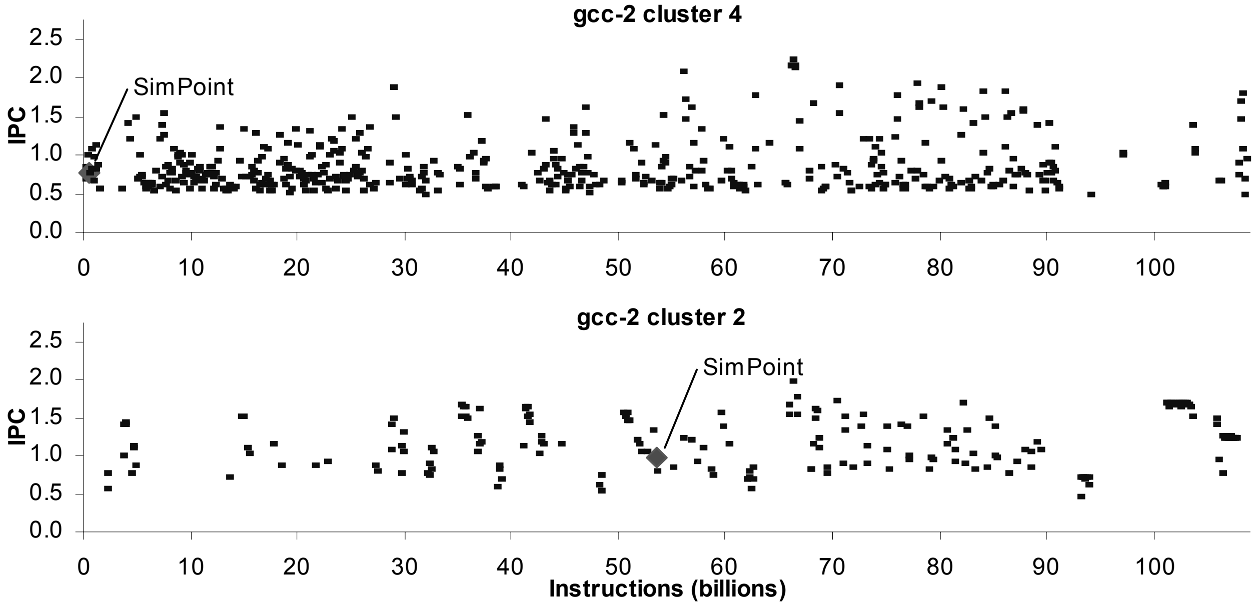


Figure 14. Cluster block IPC distribution for SimPoint.

in Figure 2). It is possible that improvements in cluster selection algorithms may indeed result in higher confidence in estimates at the cost of a larger number of units and a higher overall number of instructions measured in detail. However, such improvements must accompany careful quantitative analysis of the confidence.

Table of Contents