Statistical Sampling of Microarchitecture Simulation†

Thomas F. Wenisch Roland E. Wunderlich Babak Falsafi James C. Hoe
Computer Architecture Laboratory (CALCM)
Carnegie Mellon University, Pittsburgh, PA 15213-3890
{twenisch, rolandw, 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. The Sampling Microarchitecture Simulation (SMARTS) framework is 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 the SPEC CPU2000 benchmark suite shows that CPI can be estimated to within ± 3% with 99.7% confidence by measuring fewer than 50 million instructions per benchmark. In practice, inaccuracy in microarchitectural state initialization introduces an additional uncertainty which we empirically bound to ~2% for the tested benchmarks. We present two implementations of SMARTS that both achieve an average error of only 0.64% on CPI. SMARTSim constructs accurate model state through functional warming*—continuously warming large microarchitectural structures (e.g., caches and the branch predictor) while functionally simulating the billions of instructions between measurements—reducing average simulation turnaround from 5.5 days to 7.0 hours. TurboSMARTSim replaces functional warming with* live-points*—checkpoints that store a bare minimum of functionally-warmed state for accurate simulation of a limited execution window—further reducing average turnaround to 91 seconds.*

1. INTRODUCTION

Computer architecture research routinely employs detailed cycle-accurate simulation to explore and validate microarchitectural innovations. Despite phenomenal improvement in processor performance over the last decades, the disproportionate growth in the hardware complexity that needs to be modeled has steadily eroded simulation speed. Because of this trend, benchmark applications that are tuned to run for minutes on real hardware can require over a month to execute on today's high performance microarchitecture simulators [1,8,20].

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 papers in top-tier computer architecture conferences in 2002 presented performance claims extrapolated from abbreviated runs [27]. Unfortunately, several studies [3,7,13,15] 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.

To obtain accurate performance results representative of complete benchmarks, many proposals have advocated statistical [3,14,15] or profile-driven [7,13] 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 maintains programmer-visible architectural state, or skipped entirely by loading architectural state from checkpoints.

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 [7,9,13,14,15]. 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 [7,13,14,15], or require unrealistic assumptions about the microarchitecture (e.g., perfect branch prediction or cache hierarchies) [3].

Instead, we advocate the *Sampling Microarchitecture Simulation* (*SMARTS*) framework [27] which applies statistical sampling theory to address the shortcomings of prior simulation sampling approaches. Unlike these approaches, 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 (e.g., 10,000) of small sampling units (e.g., 1000 instructions each).

The key challenge in assessing such small sampling units lies in reconstructing accurate microarchitectural state for unbiased measurement after a checkpoint restore or an extended period of functional fast-forwarding. We have designed and implemented two alternative approaches to enable unbiased sample measurement. Our first implementation, SMARTSim, avoids measurement error from cold state by continuously warming large microarchitectural structures (e.g., caches and the branch predictor) while fast-forwarding between measurements, a warming strategy referred to as *functional warming*. Our accelerated implementation, TurboSMARTSim, replaces functional warming with *live-points*—checkpoints that store a bare minimum of functionally-warmed state for accurate simulation of a limited execution window. Live-points enable faster simulation

† This extended abstract is based on:
T. F. Wenisch, R. E. Wunderlich, B. Falsafi and J. C. Hoe "Simulation Sampling with Live-points." In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS 2006)*, March 2006.
R. E. Wunderlich, T. F. Wenisch, B. Falsafi, J. C. Hoe, "Statistical Sampling of Microarchitecture Simulation." *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, to appear.

1-4244-0054-6/06/$20.00

by eliminating all simulation time spent in functional warming—typically over 99% of SMARTSim execution time—but require certain design parameters of large microarchitectural structures to be chosen when the live-points are created.

We evaluate the SMARTS framework with both functional warming and live-points through simulators derived from SimpleScalar 3.0 `sim-outorder` [1]. Through simulation of two microarchitectures executing the SPEC CPU2000 (SPEC2K) benchmarks, we show:

- **Optimal sampling:** Both implementations achieve an actual average error of only 0.64% on CPI 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 are 174 billion instructions on average (Alpha ISA).
- **Simulation speedup:** On a 2 GHz Pentium 4, SMARTSim reduces average simulation time to 7.0 hours from 5.5 days with `sim-outorder`. SMARTSim achieves simulation speeds of over 9 MIPS. Live-point simulation sampling with TurboSMARTSim is over 250 times faster than SMARTSim (on average 91 seconds per benchmark). Although functional warming produces an aggregate of 36 TB of state while sampling SPEC2K, a gzip-compressed SPEC2K live-point library supporting 1 MB caches requires just 12 GB of storage.

This paper is organized as follows. Section 2 presents the methodology used to collect all our empirical results. We present background on the SMARTS simulation sampling framework in Section 3. In Section 4, we motivate and describe our two practical warming techniques for SMARTS: functional warming and live-points. In Section 5 we present performance analysis and results, and we describe related work in Section 6. We conclude in Section 7.

2. METHODOLOGY

We derive our implementations of the SMARTS framework from the SimpleScalar 3.0 `sim-outorder` simulator [1] for the Alpha ISA. We modify `sim-outorder`'s memory subsystem to include a store buffer and miss status holding registers (MSHRs), and model interconnect bottlenecks in the memory hierarchy. We encode live-points using ASN.1 DER format [11] and `gzip` compression, which incur minimal storage and processing time overhead. We use all 26 SPEC2K benchmarks [10] and evaluate all reference inputs except *vpr-place* and three *perlbmk* inputs, as these inputs fail to simulate correctly in `sim-outorder`. Overall, we include 41 benchmark/input set combinations in this study.

Without loss of generality, we use CPI (cycles-per-instruction) as our target metric for estimation. Simulation sampling, however, has been shown to be applicable to other performance metrics of choice [27]. We measure CPI bias by averaging actual error (relative to full `sim-outorder` simulations) over five different samples, according to the methodology described in [27].

We evaluate two microarchitectural configurations. Our baseline 8-way out-of-order superscalar model represents a processor in the current technology generation. The 16-way out-of-order superscalar configuration is included to reflect an aggressive future design point. This configuration has a wider datapath, larger out-of-order window, and larger caches, to exercise the effects of enlarged microarchitectural state. The details of the 8-way and 16-way configurations are summarized in Table 1.

Table 1. Microarchitectural configurations.

Parameter	8-way (baseline)	16-way
RUU/LSQ size	128/64	256/128
Memory system	32KB 2-way L1I/D 2 ports, 8 MSHRs 1MB 4-way L2 16-entry store buffer	64KB 2-way L1I/D 4 ports, 16 MSHRs 4MB 8-way L2 32-entry store buffer
L1/L2 line size	32/128 bytes	32/128 bytes
L1/L2/mem latency	1/12/100 cycles	2/16/100 cycles
ITLB/DTLB	4-way 128 entries/ 4-way 256 entries 200 cycle miss	4-way 128 entries/ 4-way 256 entries 200 cycle miss
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. THE SMARTS FRAMEWORK

SMARTS [27] applies statistical sampling theory [16] to find a minimal, but representative, sample of a target workload to accurately estimate performance metrics. Sampling theory further provides means to quantify the confidence in the estimated results. In our experience, the CPI (cycles per instruction) of SPEC2K benchmarks on a 8-way superscalar processor can be estimated accurately to ±3% error with 99.7% confidence from measurements of only a tiny fraction (in the range of one hundredth of one percent) of the workload.

The most fundamental theorem of sampling theory is that the sample size that must be measured to achieve a chosen confidence in estimation depends only on the target metric's coefficient of variation (standard deviation divided by mean; denoted as *CV*). Specifically, the relationship between sample size, target confidence, and *CV* is $n = ((z \cdot CV)/\varepsilon)^2$. In simulation sampling, sample size, *n*, refers to the number of measurements taken over the course of the workload, where each individual measurement is taken over a unit of *U* contiguous instructions. The quantities *z* and ε in the equation describe the target confidence and confidence interval. For instance, to achieve 99.7% confidence of ±3% error, $z = 3$ and $\varepsilon = 0.03$. In statistics terms, confidence is the probability of the estimate falling within a given interval around the true value.

Through careful application of this basic sampling theory, SMARTS simulation sampling accomplishes the following:

- **Minimal representative subset for detailed measurement.** A key insight in SMARTS's application of sampling theory is in understanding the relationship between *CV* and the sampling unit size *U*. As *U* increases, short-term fluctuations of the performance metric within a single measurement unit are averaged away. Thus, the resulting *CV* across units reflects only the long-term variation of the target metric. This leads to a lower *CV*, and therefore lower *n*. In other words, increasing *U* trades off larger measurements for fewer measurements. Our empirical study (presented later) further shows that when *U* is very small, *CV* and *n* are more sensitive to changing *U*. SMARTS can exploit this rela-

tionship to choose an optimal U to minimize the number of instructions ($n \cdot U$) in the sample selected for detailed measurement. No prior simulation sampling proposal has achieved minimal measurement.

- **Error bound and confidence in estimates.** SMARTS accompanies each estimated metric with a confidence interval describing the amount of uncertainty in the result. These confidence intervals are computed by collecting the *CV* of target metrics, while performing a simulation sampling experiment. By providing confidence intervals with results, SMARTS ensures that the measured subset is representative with respect to the chosen target metrics. These confidence intervals can also be used when comparing results across experiments to ensure the measured differences are statistically significant. Few studies presented in recent computer architecture conferences report confidence intervals or demonstrate the statistical significance of their results.
- **Exact procedure for extracting framework parameters.** SMARTS provides an exact procedure for determining the correct sample size n to achieve any desired confidence interval for a new experiment on a new simulated hardware model or benchmark workload. This procedure requires only sampled simulations, instead of complete simulations with a cycle-accurate model or collection of profiling data. Typically, the desired confidence interval can be attained with no more than two iterations of simulation sampling. Specifically, in an initial simulation run, researchers choose a likely n, based on our empirical analysis of the behavior of *CV*, to achieve a target confidence. In the exceptional scenario where the chosen sample size is later proved insufficient for the hardware model or workload, SMARTS will report a correspondingly low confidence in the simulation results. The resulting *CV* from the first simulation is substituted back into the equation above to re-calculate a better choice of n to repeat the simulation.
- **Further sample size reduction for comparative studies.** In comparative studies, researchers are often more interested in the relative performance of two designs than absolute performance. We can take advantage of this observation through a sampling procedure called *matched-pair comparison* [6]. Matched-pair comparison exploits the phenomenon that the change in performance from design x to design y tends to vary less than the absolute performance of either design. As a result, the change in performance can be assessed to a given confidence with a sample up to an order of magnitude smaller than is required for an absolute performance estimate. When applying the SMARTS framework with live-points to a comparative study, the sample size reduction from matched-pair comparison proportionally reduces total simulation turnaround time.

SMARTS technique overview. 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. Alternatively, the functional mode can be replaced with architectural state loaded from checkpoints prepared with a prior functional simulation. SMARTS alternates between detailed simulation and functional simulation to sample CPI systematically at a fixed interval. SMARTS uses systematic sampling rather than random sampling because systematic sampling is more straight-forward to implement in execution-driven simulators. For systematic sampling

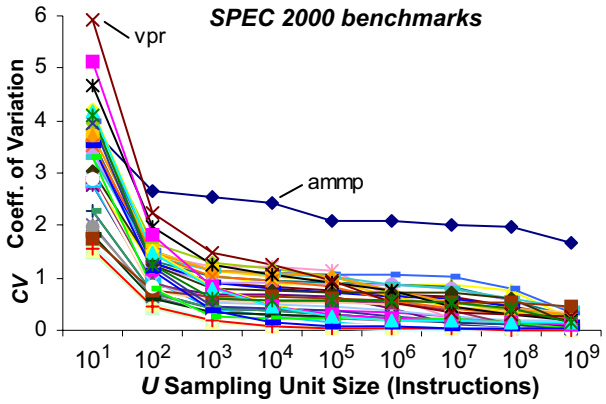

Figure 1. Coefficient of variation of CPI.

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.

Empirical results. Figure 1 shows the results of our analysis of the relationship between U and the *CV* when estimating CPI for SPEC2K benchmarks on our 8-way microarchitecture. As the graph shows, the *CV* curves for all SPEC2K benchmarks share a similar shape. Initially, there is a steep downward slope. Then, at a pivotal value of about 1000 instructions per unit, a majority of the short-term CPI fluctuation is captured within the unit and the *CV* curves flatten. Consequently, unit sizes around 1000 instructions result in minimal measurement. At U=1000, the *CV* values cluster around 1.0. The equation relating sample size to *CV* would suggest n=10,000 as a good initial guess to achieve 99.7% ±3% confidence interval for all SPEC2K benchmarks. In other words, SMARTS can estimate of CPI for SPEC benchmarks to within 3% with 99.7% confidence by measuring in detail only about 10 million instructions per benchmark.

4. SMARTS IN PRACTICE

Although statistics provides us with probabilistic guarantees that estimated results are representative, these guarantees do not assure us that estimated results are error-free. Errors introduced into the individual measurements that make up a sample (e.g., by the measurement methodology) are referred to as *bias,* and are not accounted for by statistical confidence calculations. In simulation sampling, the most common cause of bias is the cold-start effect of unwarmed microarchitectural structures. For example, assuming empty caches may result in incorrectly low performance estimates.

The primary challenge in simulation sampling is to devise a strategy to construct accurate initial state rapidly. For each measurement, the simulator must construct both architectural state (e.g., register and memory values) and microarchitectural state (e.g., pipeline components and the cache hierarchy) to avoid cold-start bias. In the following sections, we motivate and develop two simulation sampling warming strategies, functional warming and live-points.

4.1 SMARTS with Functional Warming

The cold-start 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 [14]. We refer to this solution as *detailed warming.* Figure 2

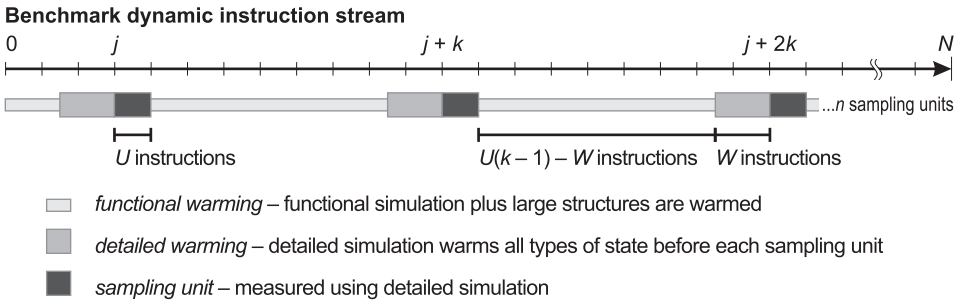

Figure 2. Systematic sampling as performed in SMARTS. Two modes of simulation are used: functional simulation, and detailed simulation. The need to determine warmup requirements for large structures, such as caches, is eliminated by performing continuous functional warming.

graphically illustrates 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.

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.

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 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 2 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 2 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.

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

Table 2. Detailed warming requirements without functional warming. (8-way)

W to achieve < 1.5% bias	Benchmarks
$W \le 50 \times 10^3$	applu, apsi, art-1, art-2, eon-1, eon-2, equake, fma3d, gzip-1, gzip-2, gzip-3, gzip-4, lucas, mesa, sixtrack, twolf
$W \le 250 \times 10^3$	crafty, eon-3, gap, gcc-1, gcc-3, gcc-4, mcf, swim, vortex-3, vpr
$W \le 500 \times 10^3$	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

Table 3. 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%

behavior does not occur; all the 8-way results presented in this article were achieved with only 2000 instructions of detailed warming, and 16-way results with 4000.

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 3 reports the residual bias in the CPI estimated by SMARTSim when functional warming is employed in conjunction with detailed warming of the aforementioned values of *W*. Benchmarks are presented in sorted-order by the worst 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.

4.2 SMARTS with Live-points

Unfortunately, as proposed, functional warming is a performance bottleneck in simulation sampling [27]. Given typical cycle-accurate simulation models (e.g., SimpleScalar `sim-outorder` [1]), the performance measurement of a wide-issue out-of-order superscalar processor using functional warming requires little detailed simulation: typically about a minute on a modern host machine. However, total runtime is orders of magnitude longer because the functional warming between detailed windows dominates runtime, occupying more than 99% of simulation runtime. Functional warming dominates simulation time because the entire benchmark's execution must be functionally simulated, even though only a tiny fraction of the execution is simulated using detailed microarchitecture timing models.

Functional warming repeats architectural state updates across different simulations of the same benchmark. Frequently, microarchitectural state updates are also identical across runs. Checkpoints can memoize the redundant calculation across runs, amortizing the one-time cost of computing warmed state.

Although modern computer architecture simulators frequently provide checkpoint creation and loading capabilities [1,8], current checkpoint implementations: (1) do not provide complete microarchitectural model state, and (2) cannot scale to the required checkpoint library size (~10,000 checkpoints per benchmark) because of multi-terabyte storage requirements.

We address the first limitation of conventional checkpoints by storing selected microarchitectural state in live-points, an approach we call *checkpointed warming*. The key challenge of checkpointed warming lies in storing microarchitectural state such that live-points can still simulate the range of microarchitectural configurations of interest. However, just as with functional warming, we can employ a brief period of detailed warming to reconstruct state for the vast majority of microarchitectural structures. By warming most structures dynamically, we avoid storing any state for these structures, and do not constrain model parameters that affect this state.

We reduce the size of conventional checkpoints by three orders of magnitude through storing in live-points only the subset of state necessary for limited execution windows, an approach we call *live-state*. Live-state exploits the brevity of SMARTS sampling units (thousands of instructions) to omit the vast majority of state.

Checkpointed warming. The key concern in evaluating checkpointed warming is the reusability of a set of checkpoints across a series of experiments. Because checkpointed warming uses a full-warming simulation to generate microarchitectural state for large structures, its accuracy is identical to full warming. When the generated live-points can be used for at least two experiments, checkpointed warming provides a net speed gain over full warming.

To maximize the reusability of live-points, we wish to place as few constraints as possible on microarchitectural configuration. Checkpointed warming dynamically reconstructs the vast majority of microarchitectural structures (e.g., queues, ROB, etc.) through detailed warming. As such, the configurations of these dynamically-warmed structures are not constrained. For the remaining few structures, for which detailed warming requirements are large or cannot be determined (e.g., caches and branch predictors), we store a representation of the structure in each live-point. The reusability of a live-point library is limited by the flexibility of these representations.

There are two basic approaches to increasing live-point reusability. First, we can collect state snapshots for multiple component configurations in a single creation pass. The second, preferable approach is to modify the saved representation such that a range of organizations can be reconstructed when a live-point is loaded. However, we cannot easily apply this adaptable approach to some structures, such as modern branch predictors, and so we must store multiple warmed configurations. Cache-like structures, including the TLB, can typically be stored using adaptable data structures.

Implementing checkpointed warming. Current publicly-available computer architecture simulators already provide a checkpoint creation and loading capability that allows the simulator to move to a particular program trace location in constant time [1,8]. These checkpoint implementations store only architecturally-visible system state (i.e., memory, architectural register and peripheral device state). A straightforward approach to implement checkpointed warming is to extend these existing checkpoints with functionally-warmed microarchitectural state.

Unfortunately, this straightforward approach is not practical because conventional checkpoints require prohibitive storage, proportional to the total memory footprint of an application (up to 200MB for SPEC2K [10]). We measured an average SPEC2K memory footprint of 105 MB. Thus, for SMARTS samples (~10,000 measurements), conventional checkpoints for all of SPEC2K require 33 TB of storage (7.2 TB with `gzip` compression). With these checkpoint sizes, simulations are I/O bound, and checkpointed warming can provide little, if any, speedup over functional warming.

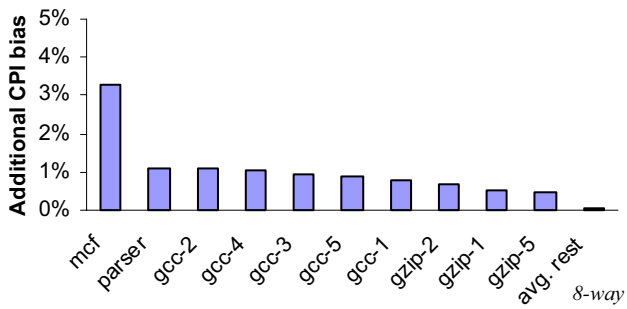

Figure 3. Restricted live-state bias. If only correct-path state is stored, wrong-path instructions are not accurately simulated.

Reducing storage with live-state. We can drastically reduce checkpoint storage cost for live-points by storing only the state that will be accessed during the brief simulation window, an approach we call *live-state.* Because the detailed windows are just a few thousand instructions, only a tiny subset of state is accessed. Simulation state that is never referenced during measurement or detailed warming can be omitted from the checkpoint without affecting the simulation.

The live-state approach stores the minimal set of accessed state for each live-point's specified simulation window. Live-points can accurately simulate only the instructions within this pre-selected window. The restriction to a pre-selected window does not impact simulation sampling because the window locations and measurement/detailed warming periods are specified in advance by the sample design.

We can identify precisely which instructions will commit during the selected window when we construct a live-point. Thus, it is straightforward to identify all the memory and microarchitectural state these instructions will access—generally less than 32 KB per live-point (uncompressed, including ASN.1 encoding overhead).

However, we cannot identify the state that is accessed on non-committed speculative paths (wrong-path instructions). It is not possible to identify *a priori* the set of wrong-path instructions that will execute in all future simulations at live-point creation time. To do so requires either fixing all simulation parameters (queue sizes and latencies), or exploring all possible speculative paths to the depth they might be followed (as bounded by, for example, ROB size). The former eliminates checkpoint reusability, while the latter requires analysis that grows exponentially with speculation depth.

Effects of wrong-path instructions. Although the effects of wrong-path instructions on the commit instruction stream are generally small [2], they cannot be ignored given our tight bias goals. Errors in wrong-path modeling cause the schedule of wrong-path execution to differ from a simulation where all state is available, which in turn perturbs the execution schedule of the commit instruction stream.

We measure the bias introduced if we restrict live-state to contain only state accessed by correct path instructions. With restricted live-state, we omit all architectural state (memory values) and microarchitectural state (cache tags and branch predictor entries) that are not accessed in the simulation window during live-point creation, leaving this state uninitialized (effectively random). A live-point with restricted live-state contains the smallest possible subset of state that can still simulate correct-path instructions (but will not accurately simulate wrong-path). Although the average bias increase for CPI is only 0.1%, the worst case is 3.3%. Figure 3 shows the bias results for the benchmarks with the most error.

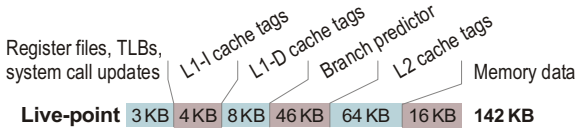

Figure 4. Breakdown of a typical live-point (uncompressed). For comparison, a conventional checkpoint is 105 MB on average.

Wrong-path instructions interact with the commit stream through resource contention and in the cache tag arrays. In the vast majority of cases, we can use branch predictor outcomes to identify the wrong-path instruction sequence, and cache tag arrays to identify wrong-path load latency. This information is sufficient to identify contention and cache tag array updates arising from speculative execution, without the need for the values accessed by wrong-path loads.

In our live-state approach, we include the microarchitectural state necessary to reflect wrong-path effects (branch predictor, cache tag arrays, TLBs), but omit memory values unless they are accessed on the correct-path. Figure 4 illustrates the contents and storage breakdown of an average live-point. By omitting the vast majority of memory values, the live-state approach reduces storage requirements from over 100 MB to 142 KB per live-point (uncompressed; assuming cache hierarchy and branch predictor of our 8-way baseline). Under this approach, unavailable memory values enter the microarchitecture (via a wrong-path load) on average less frequently than once per detailed window. We measured no appreciable increase, < 0.1% difference, in CPI bias over full warming.

Comparison to functional warming. Unlike functional warming, live-point simulation time is directly proportional to sample size. Sample size depends only on a processor's performance variability across a benchmark's execution, and the desired statistical confidence [17,27]. Hence, live-point simulation turnaround time will not increase with benchmark length.

The drawback of live-points is that they impose limits on some aspects of the simulated microarchitectural parameters (e.g., the maximum size or associativity of a cache), which constrains live-point reusability. Reusability is important because we must amortize the one-time cost of live-point creation (roughly the cost of a functional warming simulation) over a series of experiments.

5. RESULTS

In this section, we compare the performance of functional warming, live-points, and non-sampled simulation. We use each SMARTS implementation to estimate the absolute CPI of our benchmark suite. We choose sample sizes to achieve precisely 99.7% confidence of ±3% error for each result. Table 4 presents measured run-time results for each warming approach, and non-sampled runs of the complete benchmark with SimpleScalar's `sim-outorder`. We show the best, average, and worst runtimes for the two microarchitectural configurations introduced in Section 2.

Live-points eliminate the turnaround time bottleneck caused by functional warming, reducing average simulation time for SPEC2K benchmarks from 7 hours to just 1.5 minutes (8-way baseline microarchitecture). Live-point simulations often complete faster than native execution of benchmarks on our host platform, which typically requires several minutes per benchmark.

Table 4. Runtimes of SPEC2K benchmarks. We include the fastest and slowest runtimes to show the variability of each technique.

	8-way (1MB L2)					16-way (4MB L2)				
	Minimum		Average	Maximum		Minimum		Average	Maximum	
sim-outorder	2.2 h *perlbmk*	13 h *gcc-2*	**5.5 d**	15 d *mgrid*	24 d *parser*	3.8 h *perlbmk*	22 h *gcc-2*	**9.6 d**	27 d *mgrid*	42 d *parser*
Functional Warming (SMARTSim)	4.4 m *perlbmk*	29 m *gcc-2*	**7.0 h**	17 h *mgrid*	25 h *parser*	4.6 m *perlbmk*	31 m *gcc-2*	**7.3 h**	18 h *mgrid*	26 h *parser*
Live-points (TurboSMARTSim)	1 s *swim*	2 s *eon-2*	**91 s**	5.0 m *vpr*	12 m *ammp*	13 s *swim*	14 s *eon-2*	**7.6 m**	25 m *vpr*	1.3 h *ammp*

Times are specified in days (d), hours (h), minutes (m), or seconds (s).

For both SMARTSim and `sim-outorder`, simulation time varies linearly with benchmark length. Thus, we can expect simulation times to grow with longer benchmarks. In contrast, runtime with live-points depends on sample size, and thus CPI variability. We do not observe any relationship between CPI variability and benchmark length; therefore, we do not expect live-points' runtimes to increase for longer benchmarks.

Table 5 summarizes the characteristics of the warming approaches evaluated in this paper. The table shows the live-point library sizes, run times, and biases measured for each technique. Both functional warming and live-points achieve the same bias. Table 5 also indicates what microarchitecture model parameters must be fixed when live-points are created. A live-point library restricts maximum cache and TLB sizes and must include state for each branch predictor used in subsequent simulations. However, other microarchitectural configuration parameters are not fixed.

6. RELATED WORK

Many previous studies of simulation methodology present techniques orthogonal to our work. A variety of programming techniques can accelerate simulators by up to an order of magnitude without affecting simulation results [4,24]. However, simulation of complete benchmarks remains expensive. Construction and evaluation of short synthetic benchmarks with statistical properties similar to target workloads, commonly referred to as statistical simulation [18,19], can reduce simulation time to seconds. However, increasing the applicability, robustness and accuracy of these techniques remains an active research topic [5,12].

Ringenberg et al. [23] present intrinsic checkpointing, a checkpoint implementation that loads architectural state by instrumenting the simulated binary rather than through explicit simulator support. Unlike live-points, intrinsic checkpointing does not address microarchitectural state.

Our work builds upon previous work on simulation sampling. Uniform simulation sampling was first proposed in the context of trace-based cache simulation [14]. Conte et al. proposed using sampling theory to calculate confidence of performance estimates explicitly [3].

Other recent sampling proposals employ representative sampling [7,13,22]. In representative sampling, program phases are identified and a representative portion of each phase is measured. In contrast, all population elements have equal probability of inclusion in the sample under uniform sampling approaches.

The most prevalent representative sampling approach, SimPoint [7], identifies phases based on microarchitecture-independent analysis of the relative frequency of static basic blocks. Van Biesbrouck et al. [25] apply a checkpointed warming approach similar to live-points to accelerate SimPoint measurement. They report that checkpoint libraries for SimPoint-derived samples typically require less storage than high-confidence uniform samples (i.e., 99.7% confidence of ±3% error), whereas uniform samples simulate fewer instructions in detail per benchmark (~30 million rather than ~300 million instructions) and result in shorter simulation turnaround. Our experiments corroborate these results from this concurrent work. However, with uniform sampling, we can reduce turnaround time and live-point storage cost at the cost of reduced confidence. Existing representative sampling techniques do not provide quantitative measures of confidence with each result [26], and provide only a single option for runtime, storage cost, and accuracy.

Live-points have been successfully integrated into the Liberty Simulation Environment (LSE) by researchers at Princeton University [21]. LSE is a computer architecture simulation infrastructure, which models microarchitecture at a structural, rather than behavioral, level of abstraction. As such, LSE models match hardware closely, but simulation is an order of magnitude slower than `sim-outorder`. Integration of live-points into LSE reduced typical simulation times by up to 20*x* over functional warming.

Table 5. Summary of simulation sampling warming methods.

	Complete Simulation (sim-outorder)	Functional Warming (SMARTSim)	Live-points (TurboSMARTSim)
Average (worst) CPI bias	None	0.6% (1.6%)	0.6% (1.6%)
Average benchmark runtime	5.5 days	7.0 hours	91 seconds
SPEC2K checkpoint library size	N/A	N/A	12 GB (1 MB L2)
Fixed microarchitecture parameters	None	None	Max cache, TLB, branch predictors

7. CONCLUSION

To address the need for improved simulation accuracy and performance, we advocate 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 have described two alternative warming approaches that can be combined with SMARTS to provide accurate performance estimation with quantifiable statistical confidence bounds. Functional warming lends itself to easy integration into simulators that already provided functional simulation modes. We believe it is the best simulation sampling warming approach when the architectural structures under study have large warming requirements, thus making the application of live-points difficult.

Live-points reduce microarchitecture simulation time to the limit imposed by detailed simulation. Unlike functional warming, turnaround time with live-points is independent of benchmark length, depending only on the target metric's variance. Therefore, live-points enable simulation of benchmarks far longer than those used currently, with no increase in simulation time. The live-state approach enables checkpointed warming with reasonable storage requirements by storing only necessary functionally-warmed state for several thousand instructions of accurate performance simulation. A reusable live-point library for SPEC2K requires only 12 GB.

ACKNOWLEDGEMENTS

This work was partially supported by grants and equipment from IBM and Intel corporations, an NSF CAREER award, a Sloan research fellowship, and NSF grant CCR-0509356.

REFERENCES

[1] 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.

[2] 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*, Feb. 2002.

[3] 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*, Oct. 1996.

[4] M. Durbhakula, V. S. Pai, and S. Adve. Improving the accuracy vs. speed tradeoff for simulating shared-memory multprocessors with ILP processors. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, Jan. 1999.

[5] L. Eeckhout, R. B. Jr., B. Stougie, K. D. Bosschere, and L. K. John. Control flow modeling in statistical simulation for accurate and efficient processor design studies. In *Proceedings of the International Symposium on Computer Architecture*, June 2004.

[6] M. Ekman and P. Stenstrom. Enhancing multiprocessor architecture simulation speed using matched-pair comparison. In *Proc. of the Intl. Symposium on Performance Analysis of Systems and Software*, Mar. 2005.

[7] G. Hamerly, E. Perelman, J. Lau, and B. Calder. Simpoint 3.0: Faster and more flexible program analysis. *Journal on Instruction-Level Parallelism*, Sept. 2005.

[8] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk. Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture. *SIGMETRICS Performance Evaluation Review*, May 2004.

[9] J. W. Haskins and K. Skadron. Memory reference reuse latency: Accelerated warmup for sampled microarchitecture simulation. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, Mar. 2003.

[10] J. L. Henning. SPEC CPU2000: Measuring CPU performance in the new millennium. *Computer*, 33(7):28–35, 2000.

[11] International Organization for Standardization. ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). ISO/IEC 8825-1:2002 | ITU-T Rec. X.690 (2002).

[12] R. B. Jr., L. Eeckhout, and L. K. John. Debunking statistical simulation in HLS. In *Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking*, June 2004.

[13] 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*, Sept. 2000.

[14] 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):1325–1336, Feb. 1988.

[15] G. Lauterbach. Accelerating architectural simulation by parallel execution of trace samples. In *Hawaii International Conference on System Sciences*, volume Volume 1: Architecture, pages 205–210, Jan. 1994.

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

[17] W. Liu and M. Huang. Expert: Expedited simulation exploiting program behavior repetition. In *Proceedings of the International Conference on Supercomputing*, June 2004.

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

[19] 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.

[20] V. S. Pai, P. Ranganathan, and S. V. Adve. The impact of instruction-level parallelism on multiprocessors performance and simulation methodology. In *Proceedings of the Third IEEE Symposium on High-Performance Computer Architecture*, pages 72–83, February 1997.

[21] D. A. Penry, M. Vachharajani, and D. I. August. Rapid development of flexible validated processor models. TR 04-03, Princeton, Nov. 2004.

[22] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2003.

[23] J. Ringenberg, C. Peloski, D. Oehmke, and T. Mudge. Intrinsic Checkpointing: A methodology for decreasing simulation time through binary modification. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software*, Mar. 2005.

[24] E. Schnarr and J. Larus. Fast out-of-order processor simulation using memoization. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998.

[25] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Efficient sampling startup for sampled processor simulation. In *International Conference on High Performance Embedded Architectures & Compilers*, Nov. 2005.

[26] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. Hoe. An evaluation of stratified sampling of microarchitecture simulations. In *Workshop on Duplicating, Deconstructing, and Debunking, ISCA*, June 2004.

[27] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the International Symposium on Computer Architecture*, June 2003.