Memory Reference Reuse Latency: Accelerated Warmup for Sampled Microarchitecture Simulation

John W. Haskins, Jr.
Center for Computing Sciences
Bowie, MD 20715
predator@super.org

Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, VA 22904
skadron@cs.virginia.edu

***Abstract*— This paper proposes to speedup sampled microprocessor simulations by reducing warmup times without sacrificing simulation accuracy. It exploiting the observation that *of the memory references that precede a sample cluster, references that occur nearest to the cluster are more likely to be germane to the execution of the cluster itself*. Hence, while modeling all cache and branch predictor interactions that precede a sample cluster would reliably establish their state, this is overkill and leads to long-running simulations. Instead, accurately establishing simulated cache and branch predictor state can be accomplished quickly by only modeling a subset of the memory references and control-flow instructions immediately preceding a sample cluster.**

Our technique measures *memory reference reuse latencies* (MRRLs)—the number of completed instructions between consecutive references to each unique memory location—and uses these data to choose a point prior to each cluster to engage cache hierarchy and branch predictor modeling. By starting cache and branch predictor modeling late in the pre-cluster instruction stream, we were able to reduce overall simulation running times by an average of 90.62% of the maximum potential speedup (accomplished by performing no pre-cluster warmup at all), while generating an average error in IPC of less than 1%, both relative to the IPC generated by warming up all pre-cluster cache and branch predictor interactions.

I. Introduction

This paper explores a technique for accelerating sampled microarchitecture simulations by reducing the amount of cache and branch predictor warmup prior to each sample cluster where cycle-accurate simulation data are gathered. By *warmup* we refer to the practice of modeling cache and branch predictor interactions for a specified interval prior to actual data gathering, in an effort to establish the simulated cache and branch predictor state precisely as they would have appeared had the entire simulation been conducted in cycle-accurate detail.

Unfortunately, highly detailed software simulation of a microprocessor is prohibitively slow. Even on the fastest hardware, slowdowns of several orders of magnitude (relative to native execution) are common. For example, KleinOsowski *et al.* [9] show that cycle-accurate modeling of many SPEC CPU2000 [11] benchmarks on reference inputs can take many weeks. Still, software simulation is fundamental to all computer architecture research. To make simulation-driven research tractable, many studies employ *sampling*: taking measurements of a small, representative subset of the instructions that are executed over the lifetime of the benchmark. Since it is precisely the software simulation of the cycle-by-cycle progression of individual instructions through the pipeline that produces the overwhelming slowdowns, in sampled simulation only the subset of instructions which constitute the sample are modeled in cycle-accurate detail. Fortunately, measuring the instruction throughput (*i.e.*, instructions per cycle, IPC) of only a subset of the instructions can—for a properly chosen subset—yield information about the instruction throughput of a benchmark's entire end-to-end execution. Conte *et al.* [3], and Sherwood *et al.* [15], [16] propose strategies for choosing representative samples that yield very good approximations to true end-to-end IPC; both will be discussed in the Section II.

To preserve the integrity of sampled measurements, the simulated processor state must be accurately established prior to the cycle-accurate simulation of each cluster. In other words, accuracy is predicated upon successfully defeating the so-called *cold-start* bias; because cache and branch predictor performance are critical to microprocessor performance, if the state of the cache (at all levels of the hierarchy) and branch predictor do not appear at least approximately as they would have had the entire simulation been performed in cycle-accurate detail at the leading edge of a cluster, the simulation results may be inaccurate.

One straight-forward technique to guarantee the accuracy of pre-cluster cache and branch predictor state is to model the interaction of each memory reference—instructions and data—with the cache hierarchy and every control-flow instruction with the branch predictor while the simulator is executing pre-cluster instructions. (All cache and branch predictor interactions are already modeled within the cycle-accurate clusters.) In this way, the cache and branch predictor will always contain exactly the same state as if cycle-accurate simulation had been employed throughout the simulation. Though its accuracy is unimpeachable in terms of cache and branch predictor state, this *fullwarmup* method is heavy-handed. While not as expensive (in terms of running time) as cycle-accurate simulation, modeling all cache and branch predictor interactions is still

costly.

One method for further accelerating sampled simulations is to avoid *fullwarmup* by only modeling those interactions that occur within a given number of instructions prior to the cluster [3], [5], [8], [12]. Our technique makes the determination of when to engage cache and branch predictor warmup by exploiting *memory reference reuse latencies* (MRRL)—a measurement of the number of instructions that elapse between successive references to the same address. We have developed software that facilitates MRRL measurements and determines the pre-sample warmup interval independently for the instruction stream, data stream, and control-flow instruction stream.

The rest of this paper is organized as follows. We discuss related work in Section II. Section III explains Memory Reference Reuse Latency, its measurement and its significance. Section IV applies MRRL to sampled simulation. We explain our experimental methodology in Section V, present our results in Section VI and finally conclude in Section VII.

II. Related Work

Several studies examine ways to reduce overall simulation running times by executing only a small subset of the benchmark in cycle-accurate detail. Skadron *et al.* [17] identify short, representative simulation windows of 50 million instructions for the SPECInt95 benchmarks. The key insight which guides their approach is to exclude the benchmarks' unrepresentative start-up behavior (*e.g.*, data structure setup and initialization).

Conte *et al.* [3] take a different approach and instead simulate multiple randomly-chosen, fixed-sized clusters of contiguous instructions from the complete dynamic instruction stream. By choosing clusters randomly (*i.e.*, such that all parts of the dynamic instruction stream have equal probability of being selected), *random cluster sampling* is amenable to statistical analysis, and allows the determination of a confidence interval within which the true IPC resides. Their work demonstrates the applicability of random cluster sampling to microprocessor simulation and focuses on the problem of warming up the branch prediction structures (assuming a perfect cache). They furthermore show that using stale predictor state from the previous cluster plus a short warmup interval of at least 7,000 instructions [10] prior to each cluster is sufficient to minimize cold-start bias and achieve very small errors of a few percent in the mean observed IPC. In the experiments conducted for this research we use random cluster sampling, prefixing a warmup interval determined by MRRL before each cluster and preserving stale cache and branch predictor state.

Sherwood *et al.* [15] propose *Basic Block Distribution Analysis* (BBDA). Their technique profiles the execution frequency of a benchmark's basic blocks in order to isolate a contiguous subset of the dynamic instruction stream whose execution characteristics closely mimic the complete, end-to-end execution of the benchmark. BBDA's key insight is that periodic basic block execution frequency behavior reflects the periodicity of various architectural metrics such as IPC, cache miss rate, and branch predictor accuracy in cycle-accurate simulation. In [16], Sherwood *et al.* build upon the BBDA concept to create a technique that automatically isolates multiple contiguous subsets of the dynamic instruction stream since some benchmarks' behavior is too complex to be characterized by a single instruction stream slice. In both cases their aim is to reduce simulation running times by only executing in cycle-accurate detail, a small representative slice of the dynamic instruction stream.

Other heuristics for reducing cold-start bias are studied by Kessler *et al.* [8]. They consider using half of a sample's references for warmup purposes; tracking only entries that are known to contain good state; recycling stale state; and flushing state but estimating how much error this introduces.

The warmup acceleration methods proposed by [3], [8], however, may compromise the accuracy of the pre-sample state initialization if used to warmup cache state. Nguyen *et al.* [12] on the other hand, approach the problem of warmup analytically as a part of the trace-driven PARSIM parallel microprocessor simulation system. Their formula calculates a function of the cache block width, associativity, the average population density of memory references within the instruction stream, and the average steady-state cache miss ratio. This solution is a substantial improvement over previous techniques; by approaching the problem analytically, their technique is able to achieve rapid warmup without compromising accuracy.

Haskins and Skadron [5] propose a warmup acceleration technique called Minimal Subset Evaluation (MSE). The MSE technique uses formulas derived from combinatorics and probability theory to calculate for some user-chosen probability p, the number of memory references prior to each cluster that must be modeled in order to achieve accurate cache state; thus with probability p, cache state will appear exactly as it would had *fullwarmup* been used. As with PARSIM, MSE's mathematical underpinnings improve upon prior efforts by maintaining accuracy while reducing warmup times. Unlike PARSIM, MSE requires neither *a priori* knowledge of the steady-state cache miss ratio nor an instruction trace of the benchmark to be modeled (from which to derive the proportion of the instruction stream populated by memory references). The work in [5], however, only treats warmup acceleration of pre-cluster memory reference interactions with the first-level data- and instruction-cache; it is not obvious that MSE extends to (sometimes unified) secondary caches or branch predictors.

Karlsson *et al*, [6] develop an analytical model for characterizing working-set size as a function of database size in decision-support systems (DSSs). They perform an insightful investigation of temporal locality in DSSs and construct a model for identifying potentially reusable query components. Phalke and Gopinath [13] model inter-reference gaps (which are equivalent to memory reference reuse latencies) as k-th order Markov chains. By modeling per-address temporal locality in this way, they were able to develop improved algorithms for page replacement, dynamic memory management and trace compression. Thiébaut [19] draws an analogy between memory access patterns and fractal random walks on the one-dimensional lattice (where the countably infinite lattice is mimicked by a large memory address space). From

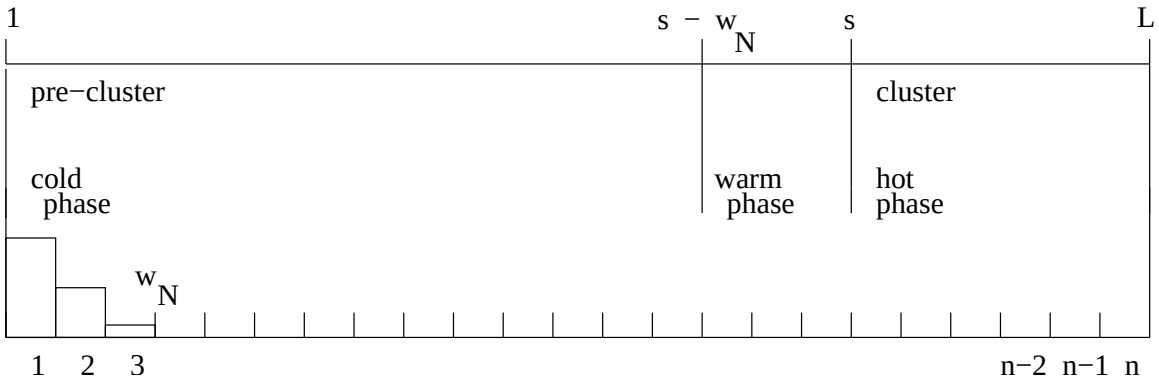

Fig. 1. Pre-cluster–cluster pair as the discrete interval $[1, L]$ partitioned into n mutually-exclusive buckets to form the $\delta insn$ histogram; here $w_N \mapsto i$ bucket$_3$, therefore warmup begins w_N instructions prior to instruction s which borders the cluster.

this framework, Thiébaut describes a method for accurately predicting the miss ratio of fully associative caches. While these works do not treat warmup in execution-driven simulation, they were instructive in their analytical assessment of temporal locality in memory reference streams.

Wood *et al.* [21] establish the concept of *cache generations*. Each cache generation begins immediately after a new line is brought into the cache and ends when the line is evicted and replaced. Their notion of cache generations establishes a framework for analytically estimating the *unknown* or *cold-start* reference miss ratio, μ. They further establish that μ is substantially higher than the miss ratio of references chosen at random. Armed with reliable $\hat{\mu}$—estimated unknown reference miss ratio—they were able to accurately estimate cache miss ratios in sampled trace-driven simulations. This research however, does not address the issue of accurately establishing simulated cache hierarchy or branch predictor state for execution-driven simulations.

In their Cache Decay research, Kaxiras *et al.* [7] propose a technique of cutting power to (heuristically presumed) dead cache lines, thereby reducing leakage power. For the SPEC CPU2000 benchmarks, their measurements show that for a 32KB L1 data-cache, a cache line's dead time can range from 45% to as much as 99% of the total time since being loaded. Their work shows that most cache lines' active lifetime is significantly longer than their useful lifetime, which confirms our hypothesis that references occurring many instructions before a cluster are unlikely to have any relevance within the cluster and can therefore be safely omitted from warmup.

As in prior research, we achieve efficient execution by breaking the simulation into three separate phases. The first, aggressive fast-forward phase can be considered the "cold" phase; this is followed by the "warm" phase, where cache and branch predictor interactions are modeled; and concluded by the "hot" phase where cycle-accurate simulation of the processor pipeline takes place. The hot phase contains sample cluster instructions and preceding cold and warm phases contain the pre-cluster instructions. Hence, for each pre-cluster–cluster pair, the aim of our research is to preserve simulation accuracy as we increase the duration of the cold phase while reducing the duration of the warm phase, always leaving the hot phase unchanged. Ad-hoc warmup methods that guess a warmup amount (*e.g.*, $X\%$ of all pre-cluster instructions) may still yield inaccurate results (if warming up only $X\%$ of pre-cluster instructions is too few) or fall short of the potential speedup (if warming up fewer than $X\%$ of pre-cluster instructions would have still yielded accurate results). By measuring the reuse latency of individual memory addresses, we were able to forge an alternative warmup acceleration technique that preserves accuracy by determining which references are likely to be germane to each cycle-accurate cluster.

III. Memory Reference Reuse Latency

Memory reference reuse latency (MRRL) refers to the elapsed time between a reference to some memory address M[A] and the next reference to M[A]. For our purposes "time" is measured in the number of completed instructions. To facilitate a rigorous discussion of MRRL, we must establish a relationship between the L instructions in a single pre-cluster–cluster pair and the elements of the discrete interval $[1, L]$; let instruction$_i \mapsto i$, for $i \in \{1, 2, ..., L\}$, as pictured in Figure 1. Imagine further, that $[1, L]$ is partitioned into $n \ll L$ mutually-exclusive buckets whose union is exactly $[1, L]$.

We measured MRRLs for each benchmark using custom-made MRRL profiling software. As the profiler simulates each pre-cluster–cluster pair, the profiling software maintains several associative arrays of memory reference addresses, M[A]—one for the instruction stream, one for the data stream, and one for the stream of branch instructions. Each element of the array is logically timestamped with the number of instructions executed as of the currently simulating memory or branch instruction; if a previously-encountered address is re-accessed, the difference of the previous timestamp and the current number of executed instructions is temporarily stored as $\delta insn$. These $\delta insn$ are used to concurrently build a reuse latency histogram by incrementing the count of the bucket that contains $\delta insn$. When a pre-cluster–cluster section concludes, the profiler outputs its $\delta insn$ histogram. These histograms contain the complete memory reference reuse latency profile

for each pre-cluster–cluster pair.

A pre-cluster–cluster histogram counts the number of references whose reuse latencies fall within n disjoint length intervals. Formally, each histogram gives the count of references for which the number of elapsed instructions between successive accesses to the same address lies within the interval subset bucket$_j$, where $j \in \{1, 2, ..., n\}$ for all n buckets. Not surprisingly, the histograms invariably tell the same story when plotted: A far greater number of references are revisited a small number of instructions after their most recent access (*i.e.*, the histogram bucket with the largest population was always bucket$_1$). Thus, the more instructions that elapse after an access to M[A], the less likely M[A] is to be accessed again during the current pre-cluster–cluster pair. This is exactly as we had expected, in light of concepts pioneered in [21] and subsequent work in [7].

From the histograms we can calculate the reuse distance corresponding to any desired *percentile* N, *i.e.*, the bucket$_j$ for which at least $N\%$ of references are contained in $\sum_{k=1}^{j}$bucket$_k$. Let $w_N \mapsto$ bucket$_j$ mean that the jth bucket of the $[1, L]$ interval is upper-bounded at $L - w_N$ instructions into the pre-cluster–cluster pair (*i.e.*, bucket$_j$: $[\alpha, L - w_N] \subset [1, L]$). In other words, of all the references in the current pre-cluster–cluster pair, $N\%$ have reuse latencies of less than w_N instructions.

By engaging warmup w_N instructions prior to the current pre-cluster–cluster boundary for large enough[1] N, we know that the overwhelming majority of addresses that will be accessed during the simulation cluster will have been initialized. We argue that if $N\%$ of references require only w_N instructions between successive accesses, then it is pointless to model the few ($(100 - N)\%$) pre-cluster cache and branch predictor interactions that occur more than w_N instructions before the cluster, since these references will probably *not* be relevant to the cluster's precision and require disproportionately long to warmup. This strategy of delaying pre-cluster cache and branch predictor modeling will be explained in more detail in the next section.

IV. Accelerating Warmup

The steps of the MRRL warmup acceleration technique are enumerated below:

1) First, the user selects the locations of the cycle-accurate clusters within the benchmark; by corollary non-cluster regions are selected simultaneously. Each cluster is paired with its own preceding non-cluster (*i.e.*, *pre-cluster*) region.
2) The user next profiles the benchmark to characterize, for each pre-cluster–cluster pair, the reuse latencies of all references that occur. As this profile data is valid for any cache and branch predictor configuration, this is a one-time cost for each benchmark sample.
3) Simulations can then be run in an aggressive fast-forward mode, updating only architected state. At w_N instructions prior to the cluster, the simulator shifts into warmup mode where cache hierarchy and branch predictor interactions are modeled. Once the cluster is reached, the cache(s) and branch predictor will contain accurate state, and cycle-accurate, simulation begins. This last step repeats for each pre-cluster–cluster pair.

[1]A discussion of "large enough" N appears in Section VI.

Pipeline	
Issue Width	8 instructions/cycle
Decode Width	8 instructions/cycle
Register Update Unit	128 entries
Load–Store Queue	32 entries
Commit Width	8 instructions/cycle
Cache Hierarchy	
L1 Data	16KB; 4-way assoc., 32B lines, 2-cycle hit
L1 Instruction	8KB; 2-way assoc., 32B lines, 2-cycle hit
L2 Unified	1MB; 4-way assoc., 64B lines, 20-cycle hit
Memory Access Latency	151 cycles
Combined Branch Predictor	
Bimodal	8192 entries
PAg	8192 entries
Return Address Stack	64 entries
Branch Target Buffer	2048 entries; 4-way assoc.
Mispredict Latency	14 cycles

TABLE I

CONFIGURATION OF SIMULATED CACHE AND BRANCH PREDICTOR.

Contrast this approach to the more conservative technique of modeling all pre-cluster cache and branch predictor interactions, *i.e.*, *fullwarmup*. Obviously, modeling all pre-cluster cache and branch predictor interactions will maintain perfect state throughout all levels of the cache hierarchy and in the branch predictor, rendering the simulation data impervious to inaccuracies that arise from cold-start bias; only sampling error remains. Reciprocally, *stale-state* or *nowarmup*—as the latter name implies—does not model any pre-cluster cache or branch predictor interactions, but merely recycles state as it appeared at the conclusion of the previous cluster. By not modeling cache and branch predictor state prior to each cluster, *nowarmup* is very susceptible to cold-start bias, as will be shown in the next Section. In our discussion of MRRL's accuracy, we refer not only to whether the true end-to-end cycle-accurate IPC is contained within a statistical confidence interval, but also to the deviation between the IPC yielded by MRRL-driven warmup, and—for the same sample—by *fullwarmup*. We measure this deviation by calculating the *relative error* thus: $100\% \cdot \frac{IPC_{MRRL} - IPC_{fullwarmup}}{IPC_{fullwarmup}}$. In our discussion of MRRL's speedup capability, we refer to the amount of potential speedup over *fullwarmup* which (as shall be shown in the next section) is the running time of *nowarmup*.

V. Methodology

The data discussed in this section were gathered using random cluster sampling as described by Conte *et al.* [3]. Random cluster sampling is attractive because its results are amenable to rigorous statistical analysis. In our experiments, 50 clusters containing 1 million instructions apiece were chosen at random from the end-to-end dynamic instruction stream. To select the clusters, benchmarks were first executed by *sim-fast*—the

benchmark	IPC_{true}	$\text{IPC}_{fullwarmup}$	$\text{IPC}_{MRRL_{0.999}}$	$\text{IPC}_{nowarmup}$
applu	0.831	-7.12%	0.00%	-0.69%
apsi	1.008	3.12%	-0.01%	-2.23%
art_110	0.598	-0.57%	0.00%	0.34%
crafty	0.569	-3.64%	-0.02%	-0.80%
equake	0.310	0.42%	0.00%	2.22%
facerec	1.446	-4.87%	0.18%	-10.46%
fma3d	0.535	-0.37%	3.90%	1.57%
galgel	1.334	-6.66%	-0.02%	-14.99%
gap	0.750	-9.61%	0.01%	-1.05%
gzip_graphic	1.365	-3.28%	-0.09%	-0.52%
lucas	0.774	2.25%	-0.04%	0.23%
mcf	0.092	3.04%	0.00%	0.84%
mgrid	0.987	4.72%	-0.01%	-1.87%
swim	0.694	3.21%	0.00%	-0.68%
twolf	0.636	-1.08%	0.13%	-1.76%
vortex_lendian2	1.057	-3.18%	0.06%	-0.63%
vpr_route	1.023	0.18%	0.00%	-1.16%
MEAN		3.37%	0.26%	2.47%

TABLE II
IPC %-ERROR RELATIVE TO FULLWARMUP ($\frac{IPC_{MRRL_{0.999}} - IPC_{fullwarmup}}{IPC_{fullwarmup}}$). MEAN CALCULATIONS BASED ON THE ABSOLUTE VALUE OF ERRORS.

rapid instruction-level simulator from the SimpleScalar [1], [2] toolset—to obtain the dynamic instruction count. Next, a simple Perl script was used to select 50 1-million-instruction clusters at random[2] from the discrete interval $[1, L]$, where L is the dynamic instruction count. The location (in the number of instructions relative to the start of execution) of the 50 clusters selected were then saved to a file, and subsequently used to drive the multiple cluster profiling and simulation steps enumerated in Section IV.

For the MRRL simulations, the warm phase was engaged w_N instructions prior to each cluster for $N = 0.999$. Recall from Section III that MRRL initiates cache and branch predictor warmup according to the maximum length of reuse latencies that compose the N-th percentile of all reuse latency measurements. We chose to study $N = 0.999$ (*i.e.*, the 99.9-th percentile) and find that it performs well in terms of absolute deviations in IPC, statistical analysis of the deviations, and speedup. Analysis of other N to find a minimum percentile is beyond the scope of this work, and is an area of future research.

All benchmarks come from the SPEC CPU2000 suite [11]; the binaries were compiled into the Alpha AXP instruction set and statically linked so that the simulations see all user-space program behavior, including library routines. The MRRL profiler and the multiple cluster simulator were adapted from *sim-safe* and *sim-outorder*, respectively, from SimpleScalar. To measure simulation time data as accurately as possible, *sim-outorder* was further modified to use the UNIX system call $getrusage()$ to monitor the CPU time of each simulation regardless of other activity on the host system. (All the scripts and software developed for this research are available for download from *http://lava.cs.virginia.edu/*.) Table I gives a brief description of the cache hierarchy and branch predictor configuration.

Once each benchmark's 50-cluster sample was selected, the next step was to profile to gather MRRL data for each benchmark. A Perl script was then used to extract w_N for each benchmark's pre-cluster–cluster pairs. When fed to the multiple cluster simulator, these data were used to demarcate the cold phase from the warm phase. The previously chosen hot phases (clusters) remained fixed just as they were during the profile.

The three metrics we use to measure MRRL's merit are percent-error IPC deviation from *fullwarmup*, accuracy with respect to the true IPC, and running time as a percentage of *fullwarmup*. For completeness and as a basis for discussing simulation acceleration, in addition to demonstrating the validity of MRRL as a tool for rapid, accurate simulation, we also show data arising from *nowarmup* for each of the three metrics aforementioned. (Recall that *nowarmup* merely recycles state from one cluster to the next, and models caching and branch prediction solely during the clusters.)

For each benchmark, Table II shows the true end-to-end IPC[3] (*i.e.*, IPC_{true}) generated by simulating in cycle-accurate detail for the entire dynamic instruction stream, *fullwarmup* IPC (*i.e.*, $\text{IPC}_{fullwarmup}$) percent-error deviation rel-

[2]By "at random," we mean such that all regions of the discrete interval $[1, L]$ have equal probability of being selected.

[3]Most of these IPCs come from the *SimPoint* [14] Web site. They were generated for a specific configuration of *sim-outorder* (linked to from the site). MRRL, *fullwarmup*, and *nowarmup* experiments compared against these IPCs use the same *sim-outorder* configuration and the same benchmark binaries.

benchmark	IPC_{true}	$IPC_{fullwarmup}$	$IPC_{MRRL_{0.999}}$	$IPC_{nowarmup}$
applu	0.831	0.772±0.151	0.772±0.151	0.767±0.158
apsi	1.008	1.039±0.063	1.039±0.064	1.039±0.064
art_110	0.597	0.595±0.029	0.595±0.029	0.597±0.029
crafty	0.569	0.548±0.014	0.548±0.014	0.544±0.014
equake	0.310	0.311±0.104	0.311±0.104	0.318±0.110
facerec	1.446	1.376±0.460	1.378±0.460	1.232±0.135
fma3d	0.535	0.533±0.061	0.554±0.058	0.542±0.055
galgel	1.334	1.245±0.339	1.245±0.339	1.059±0.327
gap	0.750	0.678±0.085	0.678±0.085	0.671±0.081
gzip_graphic	1.365	1.320±0.094	1.319±0.094	1.313±0.094
lucas	0.774	0.791±0.157	0.791±0.157	0.793±0.144
mcf	0.092	0.095±0.052	0.095±0.052	0.096±0.050
mgrid	0.987	1.034±0.106	1.034±0.106	1.014±0.080
swim	0.694	0.716±0.090	0.716±0.090	0.711±0.086
twolf	0.636	0.629±0.004	0.630±0.004	0.618±0.009
vortex_lendian2	1.057	1.023±0.040	1.024±0.040	1.017±0.040
vpr_route	1.023	1.025±0.038	1.025±0.038	1.013±0.036

TABLE III

IPC 95% CONFIDENCE INTERVALS CENTERED AROUND $\overline{IPC}$ (THE OVERALL SAMPLE IPC), FOR *fullwarmup*, $MRRL_{0.999}$, AND *nowarmup*. SUCCESSFUL SIMULATIONS CONTAIN IPC_{true} WITHIN THEIR CONFIDENCE INTERVAL.

ative to IPC_{true}, and $MRRL_{0.999}$ IPC (*i.e.*, $IPC_{MRRL_{0.999}}$) and *nowarmup* IPC (*i.e.*, $IPC_{nowarmup}$) percent-error deviation relative to $IPC_{fullwarmup}$. In other words, Table II compares the sample means for $IPC_{MRRL_{0.999}}$ and $IPC_{nowarmup}$ to the sample mean for $IPC_{fullwarmup}$, which is in turn compared to the end-to-end IPC_{true}.

VI. Evaluation

A. IPC Accuracy: MRRL versus *fullwarmup*

For most benchmarks, Table II shows that *fullwarmup*'s percent-error deviation from IPC_{true} (*i.e.*, the sampling error) is small, less than 5%. While *applu*, *galgel*, and especially *gap* buck this trend, this is not a failure of random cluster sampling, but a failure to draw a suitably large sample of clusters from the dynamic instruction stream. A larger sample (of more than 50 clusters) would have reduced the sampling error by more accurately representing all aspects of these benchmarks' behavior. Conte *et al.* [3], for example, achieve relative errors in IPC of less than 3% through sampling. Of paramount importance to our research however, is that in general, $MRRL_{0.999}$ does not introduce statistically significant *additional* nonsampling error, which arises chiefly from cold-start bias [3]. In other words, our primary objective is to develop a warmup technique such that $IPC_{MRRL_{0.999}}$ strays very little from $IPC_{fullwarmup}$, and on that count we claim victory. For all benchmarks except *fma3d*, the percent difference deviation from *fullwarmup* is less than 0.50%. *fma3d*'s seemingly drastic deviation is due to the small numbers involved in the percent-error calculation; $IPC_{fullwarmup} = 0.533$, $IPC_{MRRL_{0.999}} = 0.554$. The relative error, $100\% \cdot \left(\frac{0.554-0.533}{0.533}\right) = 3.9\%$ makes the deviation look much worse than it really is when one considers that the absolute error, $0.554 - 0.533 = 0.021$ is so small. *nowarmup* on the other hand, yields percent-error deviations from *fullwarmup* of less than 2% in general, but substantially larger relative errors for *facerec* and *galgel*. In Section VI-B, we show why these much larger errors make *nowarmup* an untrustworthy warmup strategy.

B. IPC Accuracy: MRRL versus IPC_{true}

While $MRRL_{0.999}$ is apparently a sound warmup strategy, and *nowarmup* apparently unsound, we will now rigorously demonstrate these hypotheses. As mentioned before, a significant advantage of random cluster sampling is that results obtained from this style of simulation can be statistically analyzed. Sampling produces error because only a subset of the population is measured rather than the entire population. *Random* sampling allows us to rigorously gauge the amount of error and the probability that the amount is significant, based upon the assumption that all members of the population had uniform probability of being included in the sample.

For each benchmark, the mean instruction throughput was measured by counting the number of cycles consumed by all 50 clusters. Dividing the total number of executed instructions (50 million) by this amount yielded the overall sample IPC. For a well-chosen sample, this sample IPC will be a good estimate of the end-to-end IPC. The *standard error* is a useful tool to analyze the goodness of a sample estimate [4], [18]. The standard error is computed as the quotient of the per-cluster sample standard deviation in IPC and the square root of the number of clusters:

$$s_{\overline{IPC}} = \frac{\sigma}{\sqrt{\#_{cluster}}}$$

benchmark	$t_{fullwarmup}$	$\%t_{nowarmup}$	$\%t_{MRRL_{0.999}}$	%-MPS$_{MRRL_{0.999}}$
applu	78105 sec.	59.04%	60.87%	96.90%
apsi	120925 sec.	58.11%	59.56%	97.51%
art_110	19613 sec.	35.52%	36.94%	96.00%
crafty	78906 sec.	46.52%	48.20%	96.41%
equake	54675 sec.	55.49%	57.11%	97.07%
facerec	70587 sec.	47.88%	51.98%	91.44%
fma3d	96462 sec.	59.84%	61.38%	97.43%
galgel	157325 sec.	51.63%	53.10%	97.15%
gap	82896 sec.	51.66%	53.10%	97.22%
gzip_graphic	26643 sec.	32.31%	34.58%	92.98%
lucas	46730 sec.	48.55%	50.67%	95.65%
mcf	36014 sec.	44.55%	46.45%	95.74%
mgrid	142334 sec.	58.57%	60.50%	96.71%
swim	80946 sec.	53.06%	54.81%	96.72%
twolf	133069 sec.	42.36%	44.04%	96.04%
vortex_lendian2	64839 sec.	36.71%	38.06%	96.32%
vpr_route	28358 sec.	35.66%	39.75%	88.55%
MEAN		48.09%	50.06%	90.26%

TABLE IV

MAXIMUM POTENTIAL RUNNING TIME REDUCTION (*i.e.*, $\%t_{nowarmup} = 100\% \cdot \frac{t_{nowarmup}}{t_{fullwarmup}}$) AND ACTUAL REDUCTION (*i.e.*, $\%t_{MRRL_{0.999}} = 100\% \cdot \frac{t_{MRRL_{0.999}}}{t_{fullwarmup}}$), BOTH RELATIVE TO $t_{fullwarmup}$, AND PERCENTAGE OF MAXIMUM POTENTIAL SPEEDUP ACHIEVED (*i.e.*, %-MPS$= 100\% \cdot (1 - \frac{t_{MRRL_{0.999}} - t_{nowarmup}}{t_{nowarmup}})$).

We assume (as in [3]) that error is normally distributed[4]; hence, the 95% confidence interval is $\overline{IPC} \pm 1.96 s_{\overline{IPC}}$, where $\overline{IPC}$ is the sample IPC. In other words, for a well-chosen sample, we can assume $\text{IPC}_{true} \in \left[\overline{IPC} - s_{\overline{IPC}}, \overline{IPC} + s_{\overline{IPC}}\right]$ with 95% certainty.

Let $e = \text{IPC}_{true} - \overline{IPC}$; if $[\text{IPC}_{true} - e, \text{IPC}_{true} + e] \subset [\overline{IPC} - 1.96 s_{\overline{IPC}}, \overline{IPC} + 1.96 s_{\overline{IPC}}]$, then the relative error between IPC_{true} and $\overline{IPC}$ is accurately predicted by the 95% confidence interval. The relative error between $\text{IPC}_{MRRL_{0.999}}$ and IPC_{true} was predicted by every benchmark's respective 95% confidence intervals (given in Table III) except for *crafty*, *gap*, *twolf*, and *vortex_lendian2*. However, the 95% confidence interval failed to predict the relative error between $\text{IPC}_{fullwarmup}$ and IPC_{true} for these same four benchmarks! Since *fullwarmup* perfectly models *all* inter-cluster cache and branch predictor interactions (making it impervious to nonsampling error), this failure is attributable to sampling error. Perfectly mimicking *fullwarmup* in this way is further evidence that MRRL at the 99.9-th percentile does well at approximating *fullwarmup*. In other words, $\text{MRRL}_{0.999}$ does well at eliminating nonsampling error.

In contrast, consider the $\text{IPC}_{nowarmup}$ sample means of *facerec* and *galgel*. While the 95% confidence interval of *fullwarmup* and $\text{MRRL}_{0.999}$ successfully predict their relative error deviation from IPC_{true}, *nowarmup* does not. This evidence confirms our hypothesis that their respective -10.46% and -14.99% percent-error deviations from the $\text{IPC}_{fullwarmup}$ sample means are significant.

[4]The assumption of normality is safe since the samples contain 50 clusters apiece. Samples that draw 30 or fewer elements from a population would use the Student's-t distribution [18] with $\#_{cluster} - 1$ degrees of freedom.

C. Acceleration: MRRL versus *fullwarmup*

Before discussing MRRL's acceleration capability, it is important to discuss the optimality of *nowarmup*'s runtime. Since *nowarmup* does not model any pre-cluster cache or branch predictor interactions, *nowarmup* simulations have no warm phase, only cold and hot. Because the cold phase models a proper subset of the activity modeled in the warm phase, eliminating the warm phase reduces execution time to its absolute minimum under the three-phase cold–warm–hot simulation strategy described in Section III.

Since *nowarmup* simulation time is the minimum possible simulation time it also represents the per-benchmark maximum *potential* speedup from warmup. Table IV shows that these potential speedups ranged from 32.31% for *gzip_graphic* to 59.84% for *fma3d*, of each benchmark's *fullwarmup* running time: $100\% \cdot \frac{t_{nowarmup}}{t_{fullwarmup}}$. Both the $\text{MRRL}_{0.999}$ simulation time reductions relative to *fullwarmup*, and as a percentage of the maximum potential speedup ($100\% \cdot (1 - \frac{t_{MRRL_{0.999}} - t_{nowarmup}}{t_{nowarmup}})$) are given in Table IV. In other words, Table IV shows not only the $\text{MRRL}_{0.999}$'s reduction relative to *fullwarmup*, but also how close to the maximum possible speedup each $\text{MRRL}_{0.999}$ simulation was able to come (the higher the percentage the better).

$\text{MRRL}_{0.999}$'s achieved potential speedup for all benchmarks is respectable, averaging 90.62% of the maximum, and ranging

from 88.55% for *vpr_route* to 97.51% for *apsi*. These translate into running times of only 39.75% and 59.56%, respectively of the time taken to simulate via *fullwarmup*.

VII. Conclusions and Future Work

Memory reference reuse latency analysis is a useful technique that can be used to reduce the running times of sampled simulations by reducing the amount of time spent warming up simulated cache and branch predictor state during the simulation phase preceding each sample cluster. By measuring the reuse latency (in number of instructions) between consecutive accesses to each memory address, we can discover the memory reference reuse latency that corresponds to an arbitrary percentile: MRRL_N. This MRRL_N is used to determine the amount of warmup to perform during inter-cluster regions. To make simulation as rapid as possible, cold mode uses aggressive low-detail simulation, updating only architected state; in warm mode, memory reference interactions within the cache hierarchy and branch instruction interactions with the branch predictor are also modeled. At the conclusion of the warm mode, cache and branch predictor state will be accurately established, allowing the subsequent hot mode to simulate in cycle-accurate detail without imprecision arising from cold-start bias, which can reduce accuracy.

Our results show that, used in conjunction with random cluster sampling, MRRL does not compromise accuracy. For the SPEC CPU2000 benchmarks tested, the percent-error between $\text{IPC}_{fullwarmup}$ and $\text{IPC}_{MRRL_{0.999}}$ averaged less than 1%, and was shown to be statistically insignificant for $\text{MRRL}_{0.999}$ in that all but four of the benchmarks' 95% confidence intervals predicted the observed error from IPC_{true}. Additionally, the *fullwarmup* simulations of the same four benchmarks also failed to predict the observed error. Since *fullwarmup* is impervious to nonsampling error due to cold-start bias, this implies that the failure of both $\text{MRRL}_{0.999}$ and *fullwarmup* to predict the observed error is attributable to sampling error and that $\text{MRRL}_{0.999}$ accomplishes our objective of reducing nonsampling error. Thus, we conclude that MRRL at the 99.9-th percentile mimicks *fullwarmup* well. $\text{MRRL}_{0.999}$ accomplishes our second objective as well, cutting simulation times by 50.06% on average, which is 90.62% of the maximum potential speedup.

Since MRRL works by accurately establishing cache and branch predictor state, an interesting avenue for future research would be to analyze whether MRRL accurately estimates the cache miss rate and branch misprediction rate from the sample clusters. Currently under investigation is the use of hypothesis testing to demonstrate that the difference between $\text{IPC}_{fullwarmup}$ and IPC_{MRRL_N} is statistically insignificant for some MRRL percentile N. In particular, we will implement a *matched-pairs t-test*, pitting the per-cluster IPCs of each benchmark against each other for *fullwarmup* simulations and MRRL_N simulations. In preliminary experiments we computed from the matched pairs, a set of differences which were then used to calculate a t-score based on the difference of the means, the standard error of the means, and their Pearson product-moment correlation coefficient [20], thus:

$$t = \frac{\mu_X - \mu_Y}{\sqrt{\sigma_X^2 + \sigma_Y^2 - 2r_{XY}\sigma_X\sigma_Y}}$$

where $\mu_X - \mu_Y$ is the difference of the *fullwarmup* and MRRL_N means, σ_X and σ_Y are the standard errors among the *fullwarmup* and MRRL_N cluster IPCs[5], and r_{XY} is the Pearson product-moment correlation coefficient between the *fullwarmup* and MRRL_N cluster IPCs. (This is necessary because we are measuring the effects of tested warmup strategies as different "treatments" of the same sample population [20].) This is then repeated for *fullwarmup* and *nowarmup*.

At the 5% level of significance, for instance, the critical value[6] for our 50-cluster-sample experiments is 2.0096. Table V lists the t-scores of the benchmarks calculated by pairing the cluster IPCs from *fullwarmup* and $\text{MRRL}_{0.999}$, and by pairing the cluster IPCs from *fullwarmup* and *nowarmup*. These early results are very promising and quantitatively insightful. Recall from Section VI-A, the relatively large 3.9% error between $\text{IPC}_{MRRL_{0.999}}$ and $\text{IPC}_{fullwarmup}$ for *fma3d*. We qualitatively concluded that since the absolute error was very small (0.021), that the percent-error was insignificant. Table V quantitatively confirms this since the *fma3d* t-score is less than the critical value aforementioned; thus, for *fma3d*, the difference between $\text{IPC}_{MRRL_{0.999}}$ and $\text{IPC}_{fullwarmup}$ is statistically insignificant at the 5% level.

At $N = 0.999$ only two benchmarks—*twolf* and *vortex_lendian2*—fail for $\text{MRRL}_{0.999}$ at the 5% level of significance. (Contrast this to *nowarmup*, for which eleven benchmarks fail.) We point out however, that while the $\text{MRRL}_{0.999}$ t-scores for *twolf* and *vortex_lendian2* imply that the deviation from $\text{IPC}_{fullwarmup}$ is statistically significant, the absolute difference of both ($\text{IPC}_{fullwarmup} - \text{IPC}_{MRRL_{0.999}}$) is only 0.001. It remains an open question, what threshold value of N (*e.g.*, $N = 0.9999$, $N = 0.99999$, ...) is large enough to be statistically insignificant in general, and how this will affect speedup.

The MRRL profiler software and modified version of *sim-outorder* are available at the Laboratory for Computer Architecture at Virginia Web site at *http://lava.cs.virginia.edu/*.

Acknowledgments

This material is based upon work supported in part by the National Science Foundation under grant no. CCR-0082671. The authors would like to thank Prof. Tom Conte, Prof. Mircea Stan, Prof. Bradley Calder, and the anonymous reviewers for their valuable feedback and insights.

References

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

[5] For the matched pairs t-test, μ_X and μ_Y are computed differently from the sample IPC ($\overline{IPC}$) mentioned in Section VI; rather, they are computed as the mean of per-cluster IPCs. σ_X and σ_Y are computed using μ_X and μ_Y, respectively, and are therefore also different from the standard error calculations of Section VI.

[6] According to the Student's-t distribution for 49 degrees of freedom.

benchmark	t-score$_{MRRL_{0.999}}$	t-score$_{nowarmup}$
applu	0.5988	0.2417
apsi	0.9056	2.8466
art_110	1.0474	4.1805
crafty	1.3793	4.5735
equake	1.5955	1.3638
facerec	1.3834	4.0786
fma3d	1.2416	0.7473
galgel	0.7446	2.8972
gap	0.7395	4.5349
gzip_graphic	1.9597	2.0718
lucas	1.1194	0.6455
mcf	0.5961	1.4620
mgrid	1.3420	1.7249
swim	1.2727	4.5022
twolf	4.3945	2.1865
vortex_lendian2	8.4219	3.3422
vpr_route	0.9410	9.1317

TABLE V

MATCHED-PAIRS t-TEST t-SCORES MEASURING THE STATISTICAL SIGNIFICANCE OF CLUSTER DIFFERENCES BETWEEN *fullwarmup* AND MRRL$_{0.999}$, AND *fullwarmup* AND *nowarmup*. CRITICAL t-SCORE FOR 50-CLUSTER SAMPLES IS 2.0096.

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

[3] 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 Deisgn*, Oct. 1996.

[4] J. E. Freund. *Mathematical Statistics*. Prentice-Hall, Inc., 1971.

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

[6] M. Karlsson, F. Dahlgren, and P. Stenström. An analytical model of the Working-Set sizes in decision support systems. In *Proceedings of SIGMETRICS 2000*, Jun. 2000.

[7] S. Kaxiras, Z. Hu, and M. Martonosi. Cache decay: Exploiting generational behavior to reduce cache leakage power. In *Proceedings of the 28th International Symposium on Computer Architecture*, June 2001.

[8] R. E. Kessler, M. D. Hill, and D. A. Wood. A Comparison of Trace-Sampling Techniques for Multi-Megabyte Caches. Technical Report 1048, Univ. of Wisconsin-Madison Computer Sciences Dept., September 1991.

[9] AJ KleinOsowski, J. Flynn, N. Meares, and D. J. Lilja. Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research. In *Proceedings of the Third IEEE Annual Workshop on Workload Characterization*, pages 73–82, Sep. 2000.

[10] S. Laha, J. H. Patel, and R. K. Iyer. Accurate Low-Cost Methods for Performance Evaluation of Cache Memory Systems. *IEEE Trans. Computers*, 37(11):1325–1336, November 1988.

[11] Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmarks. WWW site: http://www.specbench.org/osg/cpu2000, Dec. 1999.

[12] A. Nguyen, J. Wellman, and P. Bose. PARSIM: a parallel trace-driven simulation facility for fast and accurate performance analysis studies. In *Proceedings of the International Performance Computing, and Communications Conference*, Apr. 1997.

[13] V. Phalke and B. Gopinath. An Inter-Reference gap model for temporal locality in program behavior. In *Proceedings of SIGMETRICS 1995*, Jun. 1995.

[14] T. Sherwood, B. Calder, E. Perelman, and G. Hamerly. SimPoint home page. http://www-cse.ucsd.edu/ calder/simpoint/.

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

[16] T. Sherwood, E. Perelman, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

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

[18] M. Sternstein. *Statistics*. Barron's Educational Series, Inc., 1996.

[19] D. Thiébaut. On the fractal dimension of computer programs and its application to the prediction of the cache miss ratio. *IEEE Trans. Computers*, 8(7):1012–1026, Jul. 1989.

[20] B. Underwood, C. Duncan, J. Taylor, and J. Cotton. *Elementary Statistics*. Appleton-Century-Crofts, 1954.

[21] D. A. Wood, M. D. Hill, and R. E. Kessler. A model for estimating trace-sample miss ratios. In *Proc. ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, pages 79–89, June 1991.