Memory Reference Reuse Latency: Accelerated Sampled Microarchitecture Simulation

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

UVA Computer Science Technical Report CS-2002-19

Abstract

This paper explores techniques for speeding up sampled microprocessor simulations by exploiting the observation that of the memory references that precede a sample, references that occur nearest to the sample are more likely to be germane during the sample itself. *This means that accurately warming up simulated cache and branch predictor state only requires that a subset of the memory references and control-flow instructions immediately preceding a simulation sample need to be modeled. Our technique takes measurements of the* memory reference reuse latencies *(MRRLs) and uses these data to choose a point prior to each sample to engage cache hierarchy and branch predictor modeling.*

By starting cache and branch predictor modeling late in the pre-sample instruction stream, rather than modeling cache and branch predictor interactions for all pre-sample instructions we are able to save the time cost of modeling them. This savings reduces overall simulation running times by an average of 25%*, while generating an average error in IPC of less than* 0.7%*.*

1 Introduction

This paper explores techniques for accelerating sampled microarchitecture simulations by reducing the amount of cache and branch predictor warm-up time prior to cycle-accurate samples where simulation data are gathered. By *warm-up* 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 with 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 (less than native execution) are common. For example, Lilja *et al.* [6] show that cycle-accurate modeling of many SPEC2000 [8] benchmarks on reference inputs can take over a year. 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. Only the samples' instructions are simulated in *cycle-accurate* detail, modeling the cycle-by-cycle matriculation of individual instructions through the simulated pipeline.

To preserve the integrity of the sampled measurements, simulated processor state must be accurately established prior to full-detail simulation. 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 sample, the simulation results may be inaccurate.

One straight-forward technique to guarantee the accuracy of pre-sample cache and branch predictor state is to model the interaction of each memory reference—instruction and data—with the cache hierarchy and every control-flow instruction with the branch predictor while the simulator is executing pre-sample instructions. (Each cache interaction and branch predictor interaction is already modeled during the cycle-accurate samples.) 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 "full warm-up" 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 viable method for further accelerating sampled simulations is to avoid full warm-up by only modeling those interactions that occur within a given number of instructions prior to the sample [2, 3, 5]. Our technique makes the determination of when to engage cache and branch predictor warm-up 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 warm-up 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 2. Section 3 explains Memory Reference Reuse Latency, its measurement and its significance. Section 4 applies MRRL to sampled simulation. Finally, we explain our experimental methodology and present our results in Section 5 and conclude in Section 6.

2 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.* [11] 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.* [2] take a different approach and instead simulate multiple short samples. Their work focuses on the branch prediction structures (assuming a perfect cache) and shows that using stale predictor state from the previous sample plus a short warm-up interval [7] of at least 7,000 instructions prior to the next sample 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 a similar multiple-sample simulation regime, prefixing each sample with a warm-up interval and preserving stale cache state between samples.

Other heuristics for reducing cold-start bias are studied by Kessler *et al.* [5]. They consider using half of a sample's references for warm-up purposes; tracking only entries that are known to contain good state; using stale state from the previous sample; and flushing state but estimating how much error this introduces.

The warm-up acceleration methods proposed by [2, 5], however, may compromise the accuracy of the pre-sample state initialization. Haskins *et al.* [3] propose a warm-up acceleration technique called Minimal Subset Evaluation (MSE) which exploits the observation that only the most recent memory references prior to a sample are germane to memory references that occur during the sample itself. The MSE technique uses formulas derived from combinatorics and probability principles to calculate for some user-chosen probability p, the number of memory references prior to each sample that must be modeled in order to achieve accurate pre-sample cache state; thus with probability p, cache state will appear exactly as it would had full warm-up been used. MSE's mathematical underpinnings improve upon prior efforts by maintaining accuracy while reducing warm-up times. The work in [3], however, only treats warm-up acceleration of pre-sample memory reference interactions with the first-level data- and instruction-cache and it is not obvious that MSE extends to secondary caches or branch predictors.

Sherwood *et al.* [9] propose Basic Block Distribution Analysis (BBDA). Their technique profiles frequency characteristics of a benchmark's basic blocks in order to isolate a continuous 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 frequency behavior reflects the periodicity of various architectural metrics such as IPC, cache miss rate, and branch predictor accuracy in cycle-accurate simulation. In [10], Sherwood *et al.* build upon the BBDA concept to create a technique that automatically isolates multiple contigous subsets of the dynamic instruction stream since some benchmarks' behavior is too complex to be charactarized 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.

Wood *et al.* [12] 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 measure cache miss ratios in sampled trace-driven simulations.

In their Decay Cache research, Kaxiras *et al.* [4] propose a technique of cutting power to (heuristically presumed) dead cache lines, thereby reducing leakage power. Their measurements show that for a 32KB L1 data-cache, the proportion of the cache lines' dead time ranges from 45% to as much as 99% for the SPEC2000 benchmarks. Their work shows that most cache lines' active lifetime is significantly longer than their useful lifetime.

By measuring the reuse latency of individual memory addresses, we were able to forge an alternative warm-up acceleration technique that preserves accuracy by determining which references are likely to be germane to each cycle-accurate sample.

3 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, in a mathematical framework, we must establish a relationship between the l instructions in a single pre-sample–sample 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-sample–sample 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 timestamped with the number of instructions executed as of the currently simulating memory or branch instruction; if a previously-encountered address is reaccessed, 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 each pre-sample–sample section concludes, the profiler outputs the $\delta insn$ histogram. These histograms contain the complete memory reference reuse latency profile for each pre-sample–sample pair.

Each histogram gives the count of references for which the number of elapsed instructions between successive accesses 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-sample–sample pair.

From the histograms we were able to calculate the population of an arbitrary percentile of reuse latencies, *i.e.*, the $bucket_j$ for which at least $N\%$ of references are contained in $\bigcup_{k=1}^{j} bucket_k$. 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-sample–sample pair. In other words, of all the references in the current pre-sample–sample pair, $N\%$ have reuse latencies of less than w_N instructions.

By engaging warm-up w_N instructions prior to the current pre-sample–sample boundary for large enough N[1], we have reasonable certainty that the overwhelming majority of addresses that will be accessed during the sample will have been initialized. Essentially, 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-sample cache and branch predictor interactions that occur more than w_N instructions before the sample since these references will probably *not* be accessed during the sample. This strategy of delaying pre-sample cache and branch predictor interaction modeling will be explained in more detail in the next section.

4 Accelerating Warm-up

The steps of the MRRL warm-up acceleration technique are enumerated below:

1. First, the user selects the locations of the cycle-accurate sample regions within the benchmark; by corollary non-sample regions are selected simultaneously. Each sample is paired with its own preceding non-sample (hence referred to as *pre-sample*) region.

2. The user next profiles the benchmark to characterize, for each pre-sample–sample pair, the reuse latency 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 choice of benchmark samples.

3. Simulations can then be run in an aggressive fast-forward mode, updating only architected state. At w_N instructions prior to the cycle-accurate simulation sample, the simulator shifts into warm-up mode where memory references' interactions with the cache hierarchy and branch predictor interactions are modeled. Once the sample is reached, the cache(s) and branch predictor will contain accurate state, and full-detail cycle-accurate, simulation begins. This last step repeats for each pre-sample–sample pair.

Notice the last step. Efficient execution is achieved 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. Hence, each of the first two phases models a proper subset of the activity modeled in the subsequent phase.

Contrast this approach to the more conservative technique of modeling all pre-sample cache and branch predictor interactions. Throughout the paper we shall refer to the

[1] A discussion of "large enough" N appears in Section 5.

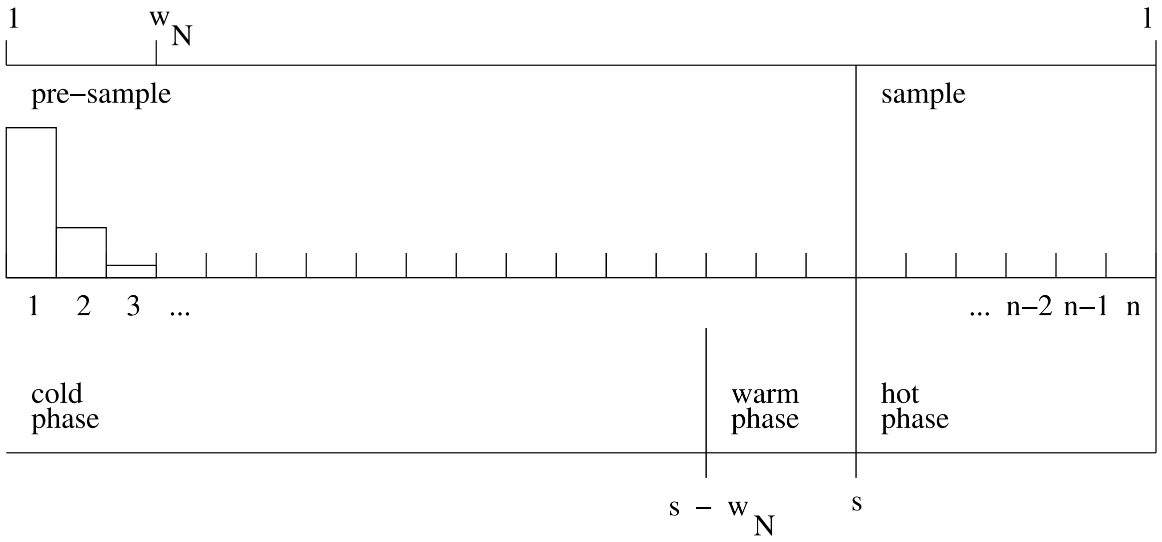

Figure 1. Pre-sample–sample pair as the discrete interval $[1, l]$ partitioned into n mutually-exclusive buckets to form the $\delta insn$ histogram; $w_N \mapsto bucket_3$, therefore warm-up begins w_N instructions prior to instruction s which borders the eminent sample.

latter as *fullwarmup*. Obviously, modeling all pre-sample 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. In our discussion of MRRL's accuracy, therefore, we refer to the data yielded by simulations that—using the technique enumerated above—model only a subset of pre-sample cache and branch predictor interactions and calculate the relative error thus: $100\% \cdot \frac{IPC_{MRRL_N} - IPC_{fullwarmup}}{IPC_{fullwarmup}}$.

Hence, for each pre-sample–sample pair, the aim of our research is to preserve accuracy as we increase the duration of the cold phase while reducing the duration of the warm phase, leaving the hot phase unchanged. While *fullwarmup* is safe, this strategy is too conservative because not all memory references that occur during the pre-sample period will impact the sample itself. Rather, as discussed in [3], we show that only the most recent references prior to a sample are relevant. Reciprocally, *nowarmup*—as its name implies—makes no effort to establish accurate state prior to each sample. Experiments that use the *nowarmup* technique do not model any pre-sample cache or branch predictor interactions. This makes *nowarmup* very susceptible to cold-start bias, as will be shown in the next Section. Other, ad-hoc warm-up methods that guess a warm-up amount (*e.g.*, 50% of all pre-sample instructions) may still yield inaccurate results (if warming up only 50% of pre-sample instructions is too few) or fall short of the potential speed-up (if warming up fewer than 50% of pre-sample instructions would have still yielded accurate results).

5 Results

The data discussed in this section were gathered using the steps enumerated in Section 4. For each pre-sample–sample pair, the warm-up phase was engaged w_N instructions prior to the sample for $N \in \{0.950, 0.990, 0.995, 0.999\}$. The benchmarks come from the SPEC2000 suite [8]; all binaries were compiled into the Alpha instruction set. The MRRL profiler and the multiple sample simulator were adapted from *sim-safe* and *sim-outorder*, respectively, from the SimpleScalar toolset version 3.0b [1].

To measure running 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 benchmark run. We also modified *sim-outorder* (and several supporting files) to incorporate enhanced statistics gathering capabilities that allowed us to engage and disengage statistics gathering at arbitrary points and for an arbitrary collection of statistics (*e.g.*, *cache_dl1*$\rightarrow$ *misses* and *pred*$\rightarrow$ addr_hits) throughout the simulation. Since the warm phase varies in size from one MRRL percentile to the next, the amount of cache and branch predictor activity also varies, but effective comparisons required comparable statistics. Since the size and locations of the full-detail samples remain fixed for a given benchmark, we decided to engage statistics gathering for the cache and branch predictor solely during the hot phases.

The two metrics we use to measure the MRRL's merit are IPC accuracy relative to *fullwarmup* and running time as a percentage of *fullwarmup*. These two metrics are tightly coupled; clearly, the latter is worthless without the former.

In other words, reducing simulation run times is only useful if we can do so while simultaneously preserving accuracy.

Our first step was to profile each of the benchmarks so that we could gather MRRL data. A simple Perl script was then used to extract w_N for each pre-sample–sample pair. These data, when fed to the multiple sample simulator were used to demarcate the cold phase from the warm phase. The previously chosen hot phase samples remained fixed just as they were during the profile.

Table 1 shows the percent error in IPC relative to *fullwarmup* using the MRRL warm-up technique and *nowarmup*. As stated in the previous Section, *nowarmup* is very susceptable to cold-start bias as is readily seen from the benchmarks *facerec*, *gcc* and especially *vpr*, and *parser*. Even though most benchmarks' *nowarmup* IPC diverges less than 1% from *fullwarmup*, the four benchmarks aforementioned effectively demonstrate that simulation data gathered using *nowarmup* cannot be trusted.

Among the benchmarks, MRRL$_N$ shows an increasing trend toward enhanced accuracy (*i.e.*, smaller relative error) as N increases. This result becomes even more compelling when one observes the trend on the four chronically inaccurate *nowarmup* experiments (*facerec*, *gcc*, *parser*, and *vpr*) highlighted previously. The only exception to the monotone increasing trend among the benchmarks is *mesa*. We speculate that this is the result of destructive interference in the branch predictor which causes *mesa* to enter its sample phases with suboptimal branch predictor state. Nevertheless, in all cases, MRRL$_{0.999}$ achieves an error of less than 0.4%; lower values of N were less reliable, but in general, still much more accurate than *nowarmup*.

Before our discussion of MRRL's ability to accelerate simulation running times, it is important to fully understand the optimality of *nowarmup*'s runtime acceleration. Since *nowarmup* does not model any cache or branch predictor interactions prior to the full-detail samples, the *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 altogether minimizes execution time to its absolute minimum under the three-phase cold–warm–hot simulation strategy described in Section 3.

Since *nowarmup* running time is the minimum possible running time[2] it also represents the per-benchmark maximum potential speed-up. Table 2 shows that these potential speed-ups ranged from 59.83% for *art* to 76.25% for *fma3d*, where these are the percentage of each benchmark's *fullwarmup* running time ($\frac{t_{nowarmup}}{t_{fullwarmup}}$). All MRRL$_N$ running time percentages shown in Table 2 give the percentage of potential *nowarmup* that each execution achieved ($100\% \cdot (1 - \frac{t_{MRRL_N} - t_{nowarmup}}{t_{nowarmup}})$). In other words, Table 2 shows how close to the maximum possible reduction each percentile was able to achieve (the higher the percentage the better).

Observe that for higher percentiles, the amount of achieved potential decreases. This, of course, is due to the fact that higher N increase the size of the warm phase relative to the cold phase, causing an ever larger number of pre-sample cache and branch predictor interactions to be modeled. In spite of this, achieved potential is still respectable, ranging from 81.54% for *vpr* to 97.07% for *lucas* at $N = 99.9\%$. These translate into running times of only 84.91% and 68.49%, respectively for each of these benchmarks relative to their *fullwarmup* running times[3]. Thus, for all benchmarks and all percentiles N, total running time was reduced by a minimum of 15%. $N = 0.995$ performs well, achieving an average error in IPC of 0.15% and an average total running time reduction of 29%; this is our recommended MRRL percentile.

6 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. By measuring the 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$_N$ then is used to mark the threshold between a simulation's cold and warm modes. 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 complications arising from cold-start bias, which can compromise accuracy.

Our results show that all benchmarks' running times were reduced by an average of 25% while generating an average error in IPC of less than 0.7%. Future research initiatives include the application of our MRRL technique to different sampling regimes. The MRRL profiler software and modified version of *sim-outorder* will soon be posted to 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.

[2]The choice of sampling strategy is beyond the scope of this work; refer to [2, 9].

[3]Percentage of *fullwarmup* $= (1 + (1 - \%_{MRRL_N})) \cdot \%_{nowarmup}$.

benchmark	IPC$_{fullwarmup}$	IPC$_{nowarmup}$	IPC$_{MRRL_{0.950}}$	IPC$_{MRRL_{0.990}}$	IPC$_{MRRL_{0.995}}$	IPC$_{MRRL_{0.999}}$
art	2.0708	-0.7920%	-0.1062%	-0.0338%	-0.0048%	0.0000%
crafty	2.3966	-0.0209%	-0.0250%	-0.0250%	-0.0250%	-0.0125%
facerec	1.6675	-2.4048%	-0.1979%	-0.1379%	-0.0960%	-0.0240%
fma3d	1.4186	-0.8248%	-0.4582%	-0.4018%	0.0141%	0.0000%
gcc	1.9937	-2.6433%	-1.2991%	-0.7323%	-0.4414%	0.0752%
gzip	2.1777	-0.1056%	-0.0413%	-0.0276%	-0.0138%	-0.0092%
lucas	0.9627	0.0831%	0.0312%	0.0208%	0.0208%	0.0208%
mesa	2.4695	0.3442%	0.3766%	0.4049%	0.3968%	0.3887%
parser	1.5248	-9.3061%	-6.4861%	-2.4200%	-1.2133%	-0.2689%
perlbmk	1.6350	1.2966%	1.3089%	1.3394%	-0.0979%	-0.0061%
twolf	1.5647	0.0000%	-0.0192%	-0.0128%	-0.0128%	-0.0128%
vortex	2.2447	-0.6593%	-0.4990%	-0.4767%	-0.3742%	-0.1114%
vpr	1.1182	-4.5698%	-2.2447%	-0.5187%	-0.2236%	-0.0179%
wupwise	1.8261	0.4600%	0.0000%	-0.0000%	0.0055%	0.0000%
MEAN		-1.4330%	-0.6900%	-0.2158%	-0.1475%	0.0016%

Table 1. IPC accuracy as %-error relative to fullwarmup ($\frac{IPC_{MRRL_N} - IPC_{fullwarmup}}{IPC_{fullwarmup}}$).

benchmark	$t_{fullwarmup}$	$\%_{nowarmup}$	$\%_{MRRL_{0.950}}$	$\%_{MRRL_{0.990}}$	$\%_{MRRL_{0.995}}$	$\%_{MRRL_{0.999}}$
art	2761	59.83%	97.46%	95.58%	95.16%	92.62%
crafty	109873	68.76%	98.35%	98.35%	98.28%	96.21%
facerec	114071	67.06%	95.23%	93.39%	91.17%	84.70%
fma3d	171281	76.25%	99.14%	98.04%	96.89%	96.24%
gcc	66856	69.80%	99.46%	98.55%	97.07%	91.23%
gzip	48673	62.37%	98.59%	96.29%	94.05%	90.86%
lucas	80891	66.54%	99.36%	98.68%	98.09%	97.07%
mesa	79019	67.37%	99.52%	98.92%	98.85%	90.00%
parser	327684	69.39%	99.77%	96.34%	92.96%	84.75%
perlbmk	16636	68.57%	99.07%	98.95%	90.72%	88.15%
twolf	213200	62.91%	97.84%	95.65%	94.20%	90.76%
vortex	77808	66.64%	99.06%	98.78%	98.16%	91.91%
vpr	57293	71.68%	99.99%	95.71%	91.77%	81.54%
wupwise	192069	69.55%	99.21%	98.91%	93.92%	87.60%
MEAN			98.72%	97.32%	95.09%	90.26%

Table 2. Maximum potential (%nowarmup) acceleration ($\frac{t_{nowarmup}}{t_{fullwarmup}}$) and acheived percentage of potential (%MRRL$_N$) running time speed-up ($100\% \cdot (1 - \frac{t_{MRRL_N} - t_{nowarmup}}{t_{nowarmup}})$)

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.