Advance Access published on May 20, 2005 doi:10.1093/comjnl/bxh103

BLRL: Accurate and Efficient Warmup for Sampled Processor Simulation

LIEVEN EECKHOUT[1], YUE LUO[2], KOEN DE BOSSCHERE[1] AND LIZY K. JOHN[2]

[1]*Department of Electronics and Information Systems, Ghent University, Belgium*
[2]*Department of Electrical and Computer Engineering, University of Texas at Austin, Austin, TX, USA*
Email: leeckhou@elis.UGent.be

Current computer architecture research relies heavily on architectural simulation to obtain insight into the cycle-level behavior of modern microarchitectures. Unfortunately, such architectural simulations are extremely time-consuming. Sampling is an often-used technique to reduce the total simulation time. This is achieved by selecting a limited number of samples from a complete benchmark execution. One important issue with sampling, however, is the unknown hardware state at the beginning of each sample. Several approaches have been proposed to address this problem by warming up the hardware state before each sample. This paper presents the boundary line reuse latency (BLRL) which is an accurate and efficient warmup strategy. BLRL considers reuse latencies (between memory references to the same memory location) that cross the boundary line between the pre-sample and the sample to compute the warmup that is required for each sample. This guarantees a nearly perfect warmup state at the beginning of a sample. Our experimental results obtained using detailed processor simulation of SPEC CPU2000 benchmarks show that BLRL significantly outperforms the previously proposed memory reference reuse latency (MRRL) warmup strategy. BLRL achieves a warmup that is only half the warmup for MRRL on average for the same level of accuracy.

Received 13 July 2004; revised 1 March 2005

1. INTRODUCTION

Current microarchitectural research relies heavily on cycle-level architectural simulations that model the execution of a benchmark on a microprocessor. Cycle-level simulations model a microarchitecture at a fairly detailed level. The price paid for such detailed simulations obviously is simulation speed. Simulating a full benchmark execution can take days or even weeks to complete. If we take into account that during microarchitectural research a multitude of design alternatives need to be evaluated, we easily end up with months or even years of simulation. As such, detailed simulation of full benchmark executions is infeasible.

Several approaches have been proposed in the recent literature to address this problem. One particular approach is sampled simulation. Sampled simulation means that a selected number of execution intervals, called samples, are simulated from a complete benchmark execution. Since the number of samples and their sizes are limited, significant simulation speedups are obtained. However, there is one particular issue that needs to be dealt with, namely the cold-start problem. The cold-start problem refers to the unknown hardware state at the beginning of each sample. An attractive solution to the cold-start problem is to simulate a number of instructions from the pre-sample without computing performance metrics. This is to warmup large hardware structures so that the hardware state at the beginning of the sample is a close estimate of the hardware state in case of detailed full benchmark simulation. Owing to an extremely long history in the microarchitectural state (e.g. in large caches), the warmup phase needs to be proportionally long. Since warm simulation can be a significant part of the total sampled simulation time, it is important to study efficient but accurate warmup strategies. Reducing the warmup length can yield significant simulation speedups.

This paper presents the boundary line reuse latency (BLRL) as a highly accurate and efficient warmup strategy. BLRL uses the reuse latencies (between two memory references accessing the same memory location) that cross the boundary line between the pre-sample and the sample to determine the warmup length per sample. By doing so, a nearly perfect warmup state is guaranteed at the beginning of each sample. We also compare BLRL with the previously proposed state-of-the-art memory reference reuse latency (MRRL) warmup strategy and conclude that BLRL significantly outperforms MRRL. Our experimental results using SPEC CPU2000 benchmarks show that BLRL

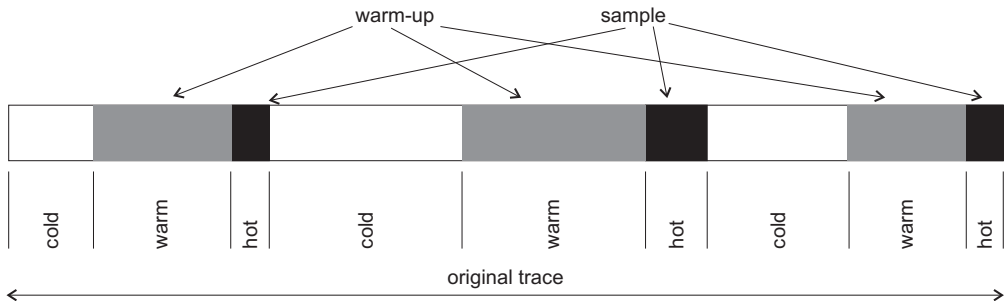

FIGURE 1. Sampled processor simulation.

achieves about half the warmup time of MRRL for the same level of accuracy to estimate the average number of cycles per instruction (CPI). This paper extends the paper published in [1] by (i) considering detailed processor simulation instead of cache simulation and (ii) comparing BLRL with MRRL. The original paper considered cache simulation and did not provide a comparison with the existing state-of-the-art.

The remainder of this paper is organized as follows. The next section gives an introduction to sampled processor simulation, after which we discuss existing warmup strategies in Section 3. Section 4 proposes our new warmup strategy, called BLRL. Section 5 details our experimental setup. In Section 6 we present and discuss our results. Finally, we conclude in Section 7.

2. SAMPLED PROCESSOR SIMULATION

In sampled processor simulation, a number of samples are chosen from a complete benchmark execution (Figure 1). The instructions between two samples are part of the pre-sample. Sampled simulation uses only the instructions in the sample to report performance results; instructions in the pre-sample are not considered.

There are basically two issues with sampling. The first issue is the selection of representative samples. The problem is to select samples in such a way that the sampled execution is an accurate picture of the complete execution of the program. As such, it is important not to limit the selection of samples to the initialization phase of the program execution. This is a manifestation of the more general observation that a program goes through various phases of execution and that the sampling should reflect this notion. In other words, samples should be chosen in such a way that all major phases are represented in the sampled execution. Several approaches have been described in the recent literature to select such samples: random sampling by Conte *et al*. [2], profile-driven sampling by scaling the basic block execution counts by Dubey and Nair [3], by selecting basic blocks with representative context information using the R-metric by Iyengar *et al*. [4, 5], periodic selection as done in SMARTS [6], selection based on clustering similarly behaving intervals as done by Lafage and Seznec [7] as well as in SimPoint [8, 9, 10].

The second issue is the correct hardware state at the beginning of each sample. This is well known in the literature as the cold-start problem. At the beginning of a sample, the correct hardware state is unknown since the instructions preceding the sample are not simulated during sampled processor simulation. Several techniques have been proposed in the literature to address this important issue (see the next section for a detailed discussion). Most of these use a number of instructions preceding the sample to warm up the hardware state before each sample. Under such a warmup strategy, sampled simulation consists of three steps (Figure 1). The first step is cold simulation in which the program execution is fast-forwarded, i.e. functional simulation without updating microarchitectural state. In case of trace-driven simulation, the instructions under cold simulation can even be discarded from the trace, i.e. need not be stored on disk. The second step is warm simulation which updates the microarchitectural state. This is typically done for large hardware structures such as caches, translation lookaside buffers and branch predictors. Under warm simulation, no performance metrics are calculated. It is important to note that the warm simulation phase can be very long since the microarchitectural state can have an extremely long history. The third step is hot simulation which includes detailed processor simulation while computing performance metrics, e.g. calculating cache and branch predictor miss rates, number of instructions retired per cycle and so on. These three steps are repeated for each sample.

Obviously, cold simulation is faster than warm simulation and warm simulation is faster than hot simulation. Austin *et al*. [11] report simulation speeds for the various simulation tools in the SimpleScalar ToolSet. They report that `sim-fast`, which corresponds to cold simulation, attains a simulation, speed of 7 million instructions per second (MIPS). Warm simulation, which is a combination of `sim-bpred` and `sim-cache`, attains $\sim$3 MIPS. Hot simulation is the slowest way of simulation with a speed of 0.3 MIPS. Owing to the fact that sampled execution only simulates a small fraction (typically $<$3%) of the complete program execution in full detail at the cycle level (under hot simulation), the total simulation time under sampled simulation is largely determined by the simulation speed under cold and warm simulation. As such, shortening the total time spent under warm simulation, i.e. trading warm simulation for cold simulation, can yield significant simulation speedups. To clarify this, we first compute the total simulation time as a function of the number of

instructions under cold, warm and hot simulation. The total simulation time T_{exec} for execution-driven simulation is proportional to

$$T_{\text{exec}} \sim f_{\text{c}} \cdot \frac{1}{7\,\text{MIPS}} + f_{\text{w}} \cdot \frac{1}{3\,\text{MIPS}} + f_{\text{h}} \cdot \frac{1}{0.3\,\text{MIPS}}, \quad (1)$$

where f_{c}, f_{w} and f_{h} are the fractions of instructions under cold, warm and hot simulation respectively. In practice, the value for f_{h} can be $>2\%$—we obtained this number from the SimPoint data (http://www.cs.ucsd.edu/∼calder/simpoint) [8, 9, 10]. If we consider the fact that a 1M instruction sample requires 10–20M warm simulation instructions on average to guarantee accurate warmup (as will be demonstrated in this paper), f_{w} then ranges from 20 to 40%. As a result, shortening the warm simulation fraction f_{w} by 50%—which is the average reduction obtained according to our results if an appropriate warmup strategy is chosen—can decrease the total simulation time by 8–15%. For trace-driven simulation, the total simulation time T_{trace} is proportional to

$$T_{\text{trace}} \sim f_{\text{w}} \cdot \frac{1}{3\,\text{MIPS}} + f_{\text{h}} \cdot \frac{1}{0.3\,\text{MIPS}}. \quad (2)$$

Reducing the warm simulation phase by 50% reduces the total simulation time by 33–50%. Because of these significant simulation time reductions, it is important to study efficient but accurate warmup techniques.

3. WARMUP STRATEGIES

3.1. Previously proposed strategies

This section gives a detailed description of previously proposed warmup strategies.

- The *cold* or *no warmup* scheme [12, 13] assumes an empty cache at the beginning of each sample. Obviously, this scheme will overestimate the cache miss rate. However, the bias can be small for large samples.
- Another option is to *checkpoint* [14] or to store the hardware state at the beginning of each sample and impose this state during sampled simulation. This approach yields a perfectly warmed up hardware state. However, the storage needed to store these checkpoints can explode in case of many samples. In addition, the hardware state needs to be stored for each specific hardware configuration. For example, for each cache and branch predictor configuration a checkpoint needs to be made. Obviously, the latter constraint implies that the complete program execution needs to be simulated for these various hardware structures.
- *Stitch* [13] approximates the hardware state at the beginning of a sample with the hardware state at the end of the previous sample.
- The *prime-xx%* method [13] assumes an empty hardware state at the beginning of each sample and uses $xx\%$ of each sample to warm up the cache. Actual simulation then starts after these $xx\%$ instructions. The warmup scheme *prime-50%* is also called *half* in the literature.
- A combination of the two previous approaches was proposed by Conte *et al*. [2]: the hardware state at the beginning of each sample is the state at the end of the previous sample plus warming up using a fraction of the sample.
- Another approach proposed by Kessler *et al*. [13, 15] is to assume an empty cache at the beginning of each sample and to estimate which cold-start misses would have missed if the cache state at the beginning of the sample was known.
- Nguyen *et al*. [16] use W instructions to warm up the cache which is calculated as follows: $W = (C/L)/(m \cdot r)$, where C is the cache capacity, L is the line size, m is the cache miss rate and r is the memory reference ratio. The problem with this approach is that the cache miss rate m is unknown; this is exactly what we are trying to approximate through sampling.
- No-state-loss (NSL) [17] scans the pre-sample and records the latest reference to each unique memory location. These references are subsequently used to warm up the caches. NSL guarantees perfect warmup for caches with least-recently used replacement. This approach was proposed in the context of sampled cache simulation; however, extending this approach to a warmup strategy for sampled processor simulation (with, e.g. branch predictor warmup) is not easily done.
- Minimal subset evaluation (MSE) proposed by Haskins and Skadron [18] determines the warmup length as follows. First, the user specifies the desired probability that the cache state at the beginning of the sample under warmup equals the cache state under perfect warmup. Second, the MSE formulas are used to determine how many unique references are required during warmup. Third, using a memory reference profile of the pre-sample it is calculated where exactly in the pre-sample the warmup should be started in order to cover these unique references.

The problem with most of these methods, except for NSL and MSE, is that they do not guarantee a (nearly) perfect hardware state at the beginning of each sample. In the following subsection we will discuss one warmup method in more detail that alleviates this problem and is considered as the current state-of-the-art in efficient and accurate warmup strategies, namely MRRL [19]. MRRL is a continuation of the work by Haskins and Skadron on MSE [18]. As stated in the introduction, we will compare our newly proposed BLRL with MRRL in more detail in Section 6. We do not compare BLRL with NSL and MSE because extending NSL to processor simulation is non-trivial and MSE was superseded by MRRL.

3.2. Memory reference reuse latency

Haskins and Skadron [19] propose MRRL for accurately warming up the hardware state at the beginning of each sample. As suggested, MRRL refers to the number of instructions between consecutive references to the same memory location, i.e. the number of instructions between

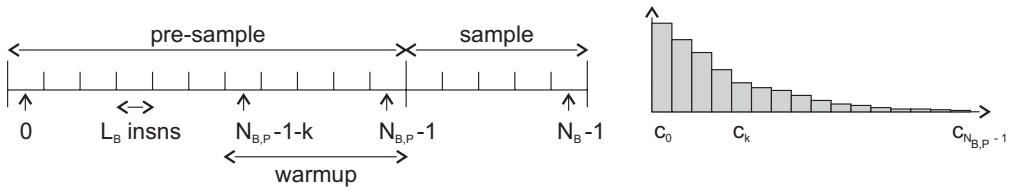

FIGURE 2. Determining warmup using MRRL.

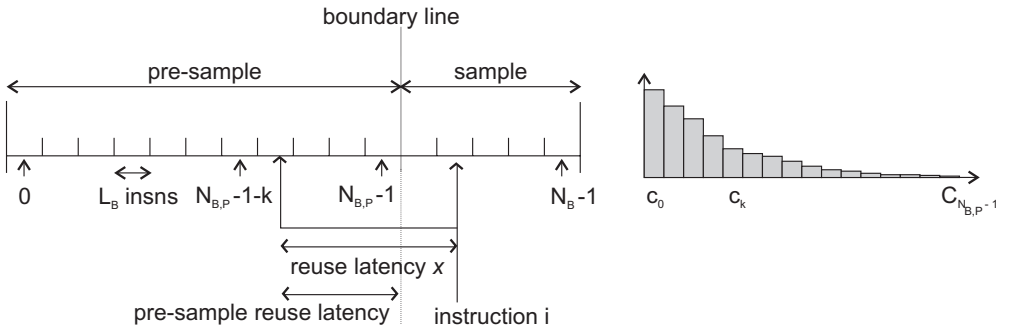

FIGURE 3. Determining warmup using BLRL.

a reference to address A and the next reference to A. For their purpose, they divide the pre-sample–sample pair into N_B non-overlapping buckets each containing L_B contiguous instructions; in other words, the total pre-sample–sample pair consists of $N_B \cdot L_B$ instructions (Figure 2). The buckets receive an index from 0 to $N_B - 1$ in which index 0 is the first bucket in the pre-sample. The first $N_{B,P}$ buckets constitute the pre-sample and the remaining $N_{B,S}$ buckets constitute the sample; obviously, $N_B = N_{B,P} + N_{B,S}$.

The MRRL warmup strategy also maintains N_B counters $c_i (0 \leq i < N_B)$. These counters, c_i, will be used to build the histogram of MRRLs. Through profiling, the MRRL is calculated for each reference and the associated counter is updated accordingly. For example, for a bucket size $L_B = 10{,}000$ (as used by Haskins and Skadron [19]) an MRRL of 124,534 will increment counter c_{12}. When the complete pre-sample–sample pair is profiled, the MRRL histogram $p_i, 0 \leq i < N_B$, is computed. This is done by dividing the bucket counters with the total number of references in the pre-sample–sample pair, i.e. $p_i = c_i / \left(\sum_{j=0}^{N_B-1} c_j\right)$. As such, $p_i = \text{Prob}[i \cdot L_B < \text{MRRL} \leq (i+1) \cdot L_B - 1]$. Not surprisingly, the largest p_is are observed for small values of i owing to the notion of temporal locality in computer program address streams. Using the histogram p_i, Haskins and Skadron calculate the bucket corresponding to a given percentile $K\%$, i.e. bucket k for which $\sum_{m=0}^{k-1} p_m < K\%$ and $\sum_{m=0}^{k} p_m \geq K\%$. This means that of all the references in the current pre-sample–sample pair, $K\%$ have a reuse latency that is smaller than $k \cdot L_B$. As such, Haskins and Skadron define these k buckets as their warmup buckets. In other words, warm simulation is started $k \cdot L_B$ instructions before the sample.

An important disadvantage of MRRL is that if there is a mismatch in the MRRL behavior in the pre-sample versus the sample, it might result in a suboptimal warmup strategy in which the warmup is either too short to be accurate or too long for the attained level of accuracy. For example, if the reuse latencies are generally larger in the sample than in the pre-sample–sample pair, the warmup will be too short and, by consequence, the accuracy might be poor. Conversely, if reuse latencies are generally shorter in the sample than in the pre-sample–sample pair, the warmup will be too long for the attained level of accuracy. One way of solving this problem is to choose a large enough percentile $K\%$. The result is that the warmup will be longer than needed for the attained accuracy.

4. BOUNDARY LINE REUSE LATENCY

BLRL is quit1e different from MRRL although it is also based on reuse latencies. In BLRL, the sample is scanned for reuse latencies that cross the pre-sample–sample boundary line, i.e. a memory location is referenced in the pre-sample and the next reference to the same memory location is in the sample. For each of these cross boundary line reuse latencies, the *pre-sample reuse latency* is calculated. This is done by subtracting the distance in the sample from the MRRL. For example, if instruction i has a cross boundary line reuse latency x, the pre-sample reuse latency then is $x - (i - N_{B,P} \cdot L_B)$ (Figure 3). A histogram is built up using these pre-sample reuse latencies. As is the case for MRRL, BLRL uses $N_{B,P}$ buckets of size L_B to limit the size of the histogram. This histogram is then normalized to the number of reuse latencies crossing the pre-sample–sample boundary line. The required warmup length is then computed to include a given percentile $K\%$ of all reuse latencies that cross the pre-sample–sample boundary line.

There are three key differences between BLRL and MRRL. First, BLRL considers reuse latencies for memory references originating from instructions in the sample only whereas MRRL considers reuse latencies for memory references

TABLE 1. The SPEC CPU2000 integer benchmarks used in this study along with their input (all inputs are reference inputs); the rightmost column shows μ_{error} under no warmup or no warm simulation during the pre-sample.

Benchmark	Input	μ_{error} (%)
vpr	route	24.05
gcc	166	7.96
crafty	ref	0.91
gap	ref	1.06
gzip	graphic	4.09
bzip2	source	19.86
vortex	lendian1	3.97
twolf	ref	2.93
eon	cook	0.22
mcf	ref	1.57

originating from instructions in both the pre-sample and sample. Second, BLRL considers only reuse latencies that cross the pre-sample–sample boundary line; MRRL considers all reuse latencies. Third, in contrast to MRRL which uses the reuse latency to update the histogram, BLRL uses the pre-sample reuse latency.

We expect BLRL to be highly accurate and efficient since it tracks the individual cross boundary line reuse latencies. These cross boundary reuse latencies in fact point to the memory locations that need to be warmed up. There is, however, one potential scenario in which BLRL will attain poor performance. Consider the case that the number of cross boundary line reuse latencies is relatively small compared with the size of the sample and that these reuse latencies have a very long pre-sample reuse latency. This will result in a long warmup; however, it will not contribute to the attained accuracy since the number of cross boundary line reuse latencies is small. As such, the warmup will be too long for the given level of accuracy. However, we expect this scenario to be rare. This is supported by the experimental results from Section 6 which show that BLRL is both more accurate and leads to shorter warmup than MRRL.

5. EXPERIMENTAL SETUP

For the evaluation we use 10 SPEC CPU2000 integer benchmarks (http://www.spec.org; Table 1). The binaries, which were compiled and optimized for the Alpha 21264 processor, are taken from the SimpleScalar website (http://www.simplescalar.com). All measurements presented in this paper are obtained using the MRRL software (http://www.cs.virginia.edu/~jwh6q/mrrl-web/) which in turn is based on the SimpleScalar software [20]. The baseline processor simulation model is given in Table 2.

In this paper we consider a sample size of 1M instructions. This sample size is in the range of sample sizes that are likely to benefit the most from efficient warmup strategies. Larger sample sizes, e.g. 100M instruction samples, do not need warmup. No warmup, i.e. only cold simulation during the pre-sample, is sufficient to faithfully estimate the performance for 100M instruction samples. Smaller sample sizes on the other hand, e.g. 1000 and 10,000 instruction samples as used in SMARTS [6], require thousands of samples to obtain accurate performance predictions. In such sampling scenarios, the pre-sample sizes are generally smaller than the observed reuse latencies. An example scenario for SMARTS uses 3000 periodically chosen 1000 instruction samples from a 100B instruction program execution. As such, the pre-sample size is ~30,000 instructions on average. The reuse latencies that we observe in this study often exceed 30,000 instructions. As such, full warmup simulation of caches and branch predictors during each pre-sample, as is done in SMARTS [6], is a practical solution for small sample sizes. Shortening this warmup could help, but the benefit of doing it is probably limited. Note that a 1M instruction sample is also the one chosen in [19] for evaluating MRRL.

We consider 50 samples (each containing 1M instructions). We select a sample for every 100M instructions. These samples were taken from the beginning of the program execution to limit the simulation time while evaluating the various warmup strategies with varying percentiles K%. Taking samples deeper down the program execution would have been too time-consuming given the large fast forwarding needed. However, we believe this does not affect the conclusions from this paper, since the warmup strategies that are evaluated in this paper can be applied to any collection of samples. Once a set of samples is provided, either warmup strategy can be applied to it.

We quantify the performance of a warmup strategy using two metrics: accuracy and warmup length. The warmup length is defined as the number of instructions under warm simulation. The accuracy is quantified as follows. We first measure the CPI for each sample under full warmup, i.e. by assuming warm simulation during the complete pre-sample. We then compute the CPI for each sample under a given warmup strategy. Using these two CPI numbers we compute the CPI prediction error on a per-sample basis. This is done as follows:

$$\text{CPI prediction error} = \frac{|\text{CPI short} - \text{CPI full}|}{\text{CPI full}}, \quad (3)$$

where CPI short and CPI full are the CPI under shortened and full warmup respectively. As such, we obtain 50 CPI prediction errors. We subsequently compute the average per-sample CPI prediction error, μ_{error}. The reason why we use average per-sample CPI prediction errors instead of aggregate CPI prediction errors—by comparing the overall CPI under shortened warmup versus full warmup—is that the latter approach might hide inaccuracies in particular samples from the aggregate CPI numbers. For example, a positive CPI prediction error in one sample can be compensated for by a negative CPI prediction error in another sample. Using the average per-sample CPI prediction error, μ_{error}, alleviates this problem.

In its rightmost column, Table 1 shows μ_{error} under the no-warmup strategy, i.e. no warm simulation during the pre-sample. These data show that warmup is clearly needed to address the cold-start problem. Note that this error is to be

TABLE 2. Baseline processor simulation model.

Instruction cache	8 KB, 2-way set-associative, 32-byte block, 1 cycle access latency
Data cache	16 KB, 4-way set-associative, 32-byte block, 2 cycles access latency
Unified L2 cache	1 MB, 4-way set-associative, 64-byte block, 30 cycles access latency
I-TLB and D-TLB	32-entry 8-way set-associative with 4 KB pages
Memory	300 cycle round trip access
Branch predictor	8 K-entry hybrid predictor selecting between an 8 K-entry bimodal predictor and a two-level (8 K $\times$ 8 K) local branch predictor xor-ing the local history with the branch's PC, 512-entry 4-way set-associative BTB and 64-entry RAS
Speculative update	At dispatch time
Branch misprediction penalty	10 cycles
IFQ	32-entry instruction fetch queue
RUU and LSQ	128 entries and 64 entries respectively
Processor width	8 issue width, 8 decode width (fetch speed = 2), 8 commit width
Functional units	8 integer ALUs, 4 load/store units, 2 fp adders, 2 integer and 2 fp mult/div units

considered on top of the sampling error. Indeed, the overall CPI error (theoretically) is the sum of the sampling error plus the error due to inaccurate warmup. Perelman *et al*. [9] report average CPI sampling errors ranging from 2 to 4%. These errors are due to sampling inaccuracies only since they assume perfect warmup in their experiments. As such, the additional error due to the cold-start problem should be small enough not to increase the overall CPI error too much.

Next to benchmark-specific information, we will also report the average numbers over all benchmarks. More specifically, we will average the warmup length and the average per-sample CPI prediction error, μ_{error} over all benchmarks. This will be done using the arithmetic average for the following reasons. For the warmup length, the arithmetic average is directly proportional to the total simulation time spent in warmup when simulating the complete benchmark suite. For the CPI prediction error, μ_{error}, the arithmetic average penalizes large inaccuracies more than the geometric average would do. For example, if one particular benchmark has a larger error than the other benchmarks, the arithmetic average error will be larger than the geometric average error. This makes sense for our purpose since we want the prediction errors for all benchmarks to be low.

6. RESULTS

Table 3 shows the results of comparing BLRL and MRRL. This is done for different values of the percentile K%. For BLRL, we use $K = 85\%, K = 90\%$ and $K = 95\%$; for MRRL, we use $K = 99.5\%$ and 99.9%. To compare the performance of a warmup strategy, we take both the warmup length and the CPI prediction error into account. We observe that BLRL performs significantly better than MRRL on average. For example, compare BLRL-90% versus MRRL-99.9%: BLRL-90% attains a higher accuracy than MRRL-99.9% ($\mu_{\text{error}} = 0.30\%$ versus 0.43% respectively) with a shorter warmup length (589M versus 896M instructions respectively). In other words, the error is reduced by 30% while having a 34% shorter warmup. Or, when comparing BLRL-85% versus MRRL-99.9%, we observe that BLRL attains the same accuracy as MRRL with a warmup length that is 49% shorter (453M versus 896M respectively). Note that although the error rate reductions seem significant in relative terms, they are not that significant in absolute terms. Haskins and Skadron [19] showed, based on statistical tests, that the per-sample CPI numbers obtained through MRRL are statistically insignificant from the per-sample CPI numbers obtained through full warmup. Therefore, the improvement in terms of accuracy demonstrated here through BLRL is to be seen within this margin of error. Therefore, we conclude that BLRL achieves a similar level of accuracy as MRRL but

TABLE 3. Comparison of BLRL versus MRRL.

	BLRL			MRRL	
	85%	90%	95%	99.5%	99.9%
Average per-sample CPI prediction error, μ_{error}					
bzip 2	1.09	0.57	0.21	1.37	1.02
crafty	0.12	0.07	0.04	0.48	0.09
eon	0.02	0.02	0.02	0.02	0.01
gap	0.20	0.11	0.11	0.61	0.40
gcc	0.34	0.27	0.25	0.49	0.42
gzip	0.18	0.13	0.06	0.17	0.09
mcf	0.05	0.05	0.05	0.08	0.08
twolf	0.14	0.09	0.05	0.56	0.14
vortex	1.57	1.39	1.27	2.74	1.92
vpr	0.55	0.29	0.18	1.36	0.12
avg	0.43	0.30	0.22	0.79	0.43
Warmup length (in millions)					
bzip2	1215	1422	1672	1253	2323
crafty	371	733	1785	24	306
eon	110	124	137	78	149
gap	71	96	109	3	69
gcc	130	189	254	135	347
gzip	285	312	338	353	425
mcf	110	145	788	1213	2119
twolf	232	328	440	383	658
vortex	402	640	1569	47	235
vpr	1609	1897	2455	945	2325
avg	453	589	955	443	896

TABLE 4. Obtained CPI prediction error reduction and warmup length reduction of BLRL-90% versus MRRL.

	BLRL			MRRL			MRRL–BLRL	
	K_{BLRL}%	μ_{error}(%)	Warmup	K_{MRRL}%	μ_{error}(%)	Warmup	μ_{error}(%)	Warmup
bzip2	90	0.57	1422	99.9	1.02	2323	0.45	901
crafty	90	0.07	733	99.95	0.05	749	−0.02	16
eon	90	0.02	124	99	0.02	19	0.00	−105
gap	90	0.11	96	99.95	0.22	101	0.11	5
gcc	90	0.27	189	99.95	0.29	461	0.02	272
gzip	90	0.13	312	99.5	0.17	353	0.04	41
mcf	90	0.05	145	99.95	0.06	2171	0.01	2026
twolf	90	0.09	328	99.95	0.11	793	0.02	465
vortex	90	1.39	640	99.96	1.39	683	0.00	43
vpr	90	0.29	1897	99.7	0.26	1452	−0.03	−445

achieves this level of accuracy with a significantly shorter warmup length—the warmup length under BLRL is nearly half the warmup length under MRRL.

Table 4 compares BLRL with a fixed percentile $K_{BLRL} = 90\%$ versus MRRL with a variable percentile K_{MRRL}% for 1M instruction samples on a per-benchmark basis. The motivation for such an analysis is to quantify how much shorter the warmup is for BLRL than for MRRL to yield the same level of accuracy. For this table, we have run a large number of experiments with varying percentiles K_{MRRL}% for MRRL. For each benchmark a different percentile K_{MRRL}% is chosen so that the the CPI prediction error for MRRL is close to that for BLRL. In its two rightmost columns, Table 4 presents the reduction in CPI prediction error (in percentage point) and warmup length (in millions of instructions). Positive values indicate that BLRL yields better accuracy and shorter warmup than MRRL. These data show that for 7 out of the 10 benchmarks, BLRL attains smaller errors and shorter warmup than MRRL. For four benchmarks, the reduction in warmup length over MRRL is very large: bzip2 (39%), gcc (59%), twolf (59%) and mcf (93%). For one benchmark, namely vpr, MRRL attains a smaller error and a shorter warmup than BLRL. For two benchmarks (crafty and eon), BLRL and MRRL are comparable.

Another way of looking at the performance of warmup strategies is to plot the average CPI prediction error, μ_{error}, versus warmup length. Figure 4 shows such a graph for four benchmarks: twolf, gcc, bzip2 and vpr. The different points for each curve correspond to different values of the percentiles K_{BLRL}% and K_{MRRL}%. Obviously, increasing percentiles K% correspond to increasing warmup lengths and decreasing CPI prediction errors. This graph shows that for twolf, gcc and bzip2, BLRL is significantly better than MRRL. We observed similar graphs for most of the other benchmarks. For vpr on the other hand, MRRL seems to outperform BLRL.

In order to understand where the (small) CPI prediction errors come from, we have performed an error analysis. For this purpose we have collected various metrics under BLRL-85% and MRRL-99.9%: the L1 I-cache miss rate, the L1 D-cache miss rate, the unified L2-cache miss rate and the branch misprediction rate (Table 5). The error rates shown in this table are absolute error rates which are computed as follows:

$$M\text{prediction error} = |M\text{short} - M\text{full}|, \qquad (4)$$

where M short and M full denote a metric M under shortened and full warmup respectively. We use this absolute error rate instead of the relative error rate since the miss rates are small numbers—a relative error rate would enlarge small differences for small numbers without providing a useful meaning. We conclude from Table 5 that the error rates for the L1 I and D caches are zero in nearly all cases. For the branch misprediction rates we observe higher error rates, but these errors are still <0.11%. The highest error rates are observed for the L2-cache miss rates. For example, for bzip2 the error rates are 2.94 and 2.28% for MRRL and BLRL respectively; for vortex the error rates are 0.86 and 0.84% for MRRL and BLRL respectively. Note that the higher error rates for the L2 caches for these two benchmarks result in higher error rates in overall CPI (Table 3).

7. CONCLUSION

Architectural simulation is an essential tool for micro-architectural research to obtain insight into the cycle-level behavior of current microprocessors. Unfortunately, these architectural simulations are extremely time-consuming, especially if industry standard benchmarks need to be simulated to completion. Sampled simulation is an often-used solution to drastically reduce the total simulation time. In sampled simulation, a well-chosen set of samples is selected such that they represent an accurate picture of the complete benchmark execution.

An important problem with sampling, however, is the unknown hardware state at the beginning of each sample. To accurately estimate this hardware state researchers have proposed various warmup strategies. This is done by simulating additional instructions from the pre-sample without computing performance metrics; this is particularly

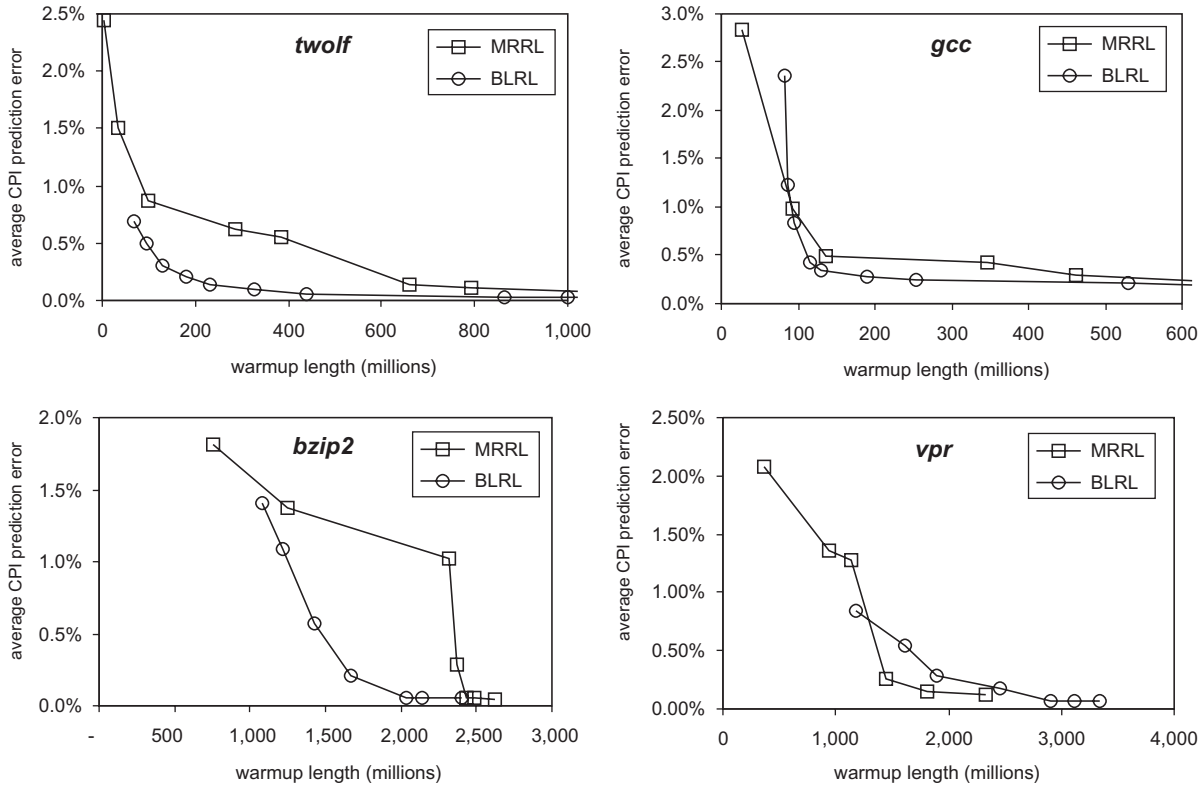

FIGURE 4. Average CPI prediction error μ_{error} versus warmup length for twolf, gcc, bzip2 and vpr.

TABLE 5. Error rates between BLRL-85 and MRRL-99.9% versus full warmup: L1 I-cache miss rate, L1 D-cache miss rate, L2-cache miss rate and branch misprediction rate.

	BLRL−85%				MRRL−99.9%			
	IL1(%)	DL1(%)	L2(%)	bpred(%)	IL1(%)	DL1(%)	L2(%)	bpred(%)
bzip2	0.00	0.00	2.28	0.01	0.00	0.00	2.94	0.01
crafty	0.01	0.00	0.03	0.03	0.01	0.00	0.02	0.03
eon	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01
gap	0.00	0.00	0.04	0.04	0.00	0.00	0.07	0.02
gcc	0.00	0.00	0.03	0.11	0.00	0.00	0.03	0.08
gzip	0.00	0.00	0.08	0.01	0.00	0.00	0.05	0.01
mcf	0.00	0.00	0.03	0.00	0.00	0.00	0.07	0.00
twolf	0.00	0.00	0.02	0.02	0.00	0.00	0.02	0.02
vortex	0.01	0.00	0.84	0.05	0.01	0.00	0.86	0.04
vpr	0.00	0.00	0.19	0.01	0.00	0.00	0.02	0.01

useful for large hardware structures such as caches and branch predictors. Since warm simulation has a significant impact on the overall sampled simulation time, it is important to study efficient but accurate warmup strategies. In this paper we proposed BLRL which uses reuse latencies (between memory references to the same memory location) that cross the boundary line between the pre-sample and the sample. BLRL uses a percentage (e.g. 90%) of these reuse latencies to calculate the warmup length per sample. This paper also compared BLRL with the previously proposed MRRL. Our experimental results using SPEC CPU2000 and detailed processor simulation showed that BLRL outperforms MRRL significantly. BLRL achieves a warmup that is nearly half the size, on average, of the warmup under MRRL for the same level of accuracy.

ACKNOWLEDGEMENTS

Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research, Flanders, Belgium (F.W.O. Vlaanderen). This research is sponsored by Ghent University and the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). This research is also partially supported by the National Science Foundation under grant number 0113105, by IBM through a CAS award and a SUR grant, and by AMD, Intel and Microsoft corporations.

REFERENCES

[1] Eeckhout, L., Eyerman, S., Callens, B. and De Bosschere, K. (2003) Accurately warmed-up trace samples for the evaluation of cache memories. In *Proc. 2003 High Performance Computing Symposium (HPC-2003)*, Orlando, FL, March 30–April 3, pp. 267–274. SCS, San Diego, CA.

[2] Conte, T. M., Hirsch, M. A. and Menezes, K. N. (1996) Reducing state loss for effective trace sampling of superscalar processors. In *Proc. 1996 International Conference on Computer Design (ICCD-96)*, Austin, TX, October 7–9, pp. 468–477. IEEE CS, Los Alamitos, CA.

[3] Dubey, P. K. and Nair, R. (1995) *Profile-driven Sampled Trace Generation*. Technical Report RC 20041, IBM Research Division, T. J. Watson Research Center.

[4] Iyengar, V. S. and Trevillyan, L. H. (1996) *Evaluation and Generation of Reduced Traces for Benchmarks*. Technical Report RC 20610, IBM Research Division, T. J. Watson Research Center, Yorktown, NY.

[5] Iyengar, V. S., Trevillyan, L. H. and Bose, P. (1996) Representative traces for processor models with infinite cache. In *Proc. Second Int. Symp. on High-Performance Computer Architecture (HPCA-2)*, San Jose, CA, February 3–7, pp. 62–73. IEEE Computer Society, Los Alamitos, CA.

[6] Wunderlich, R. E., Wenish, T. F., Falsafi, B. and Hoe, J. C. (2003) SMARTS: accelerating microarchitecture simulation via rigorous statistical sampling. In *Proc. 30th Annual Int. Symp. on Computer Architecture (ISCA-30)*, San Diego, CA, June 9–11, pp. 84–95. IEEE Computer Society, Los Alamitos, CA.

[7] Lafage, T. and Seznec, A. (2000) Choosing representative slices of program execution for microarchitecture simulations: a preliminary application to the data stream. In *Proc. IEEE 3rd Annual Workshop on Workload Characterization (WWC-2000) held in conjunction with the Int. Conf. on Computer Design (ICCD-2000)*, Austin, TX, September 16, Kluwer International Series in Engineering and Computer Science Series Archive. Kluwer, Norwell, MA, pp. 145–163.

[8] Sherwood, T., Perelman, E., Hamerly, G. and Calder, B. (2002) Automatically characterizing large scale program behavior. In *Proc. Tenth Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, October 5–9, pp. 45–57. ACM, New York, NY.

[9] Perelman, E., Hamerly, G. and Calder, B. (2003) Picking statistically valid and early simulation points. In *Proc. 12th Int. Conf. on Parallel Architectures and Compilation Techniques (PACT-2003)*, New Orleans, LA, September 27–October 1, pp. 244–256. IEEE Computer Society, Los Alamitos, CA.

[10] Sherwood, T., Perelman, E. and Calder, B. (2001) Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proc. 2001 Int. Conf. on Parallel Architectures and Compilation Techniques (PACT-2001)*, Barcelona, Spain, September10–12, pp. 3–14. IEEE Computer Society, Los Alamitos, CA.

[11] Austin, T., Larson, E. and Ernst, D. (2002) SimpleScalar: an infrastructure for computer system modeling. *IEEE Computer*, **35**, 59–67.

[12] Crowley, P. and Baer, J.-L. (1999) Trace sampling for desktop applications on Windows NT. In *Workload Characterization: Methodology and Case Studies*. IEEE Computer Society Press.

[13] Kessler, R. E., Hill, M. D. and Wood, D. A. (1994) A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Trans. Comput.*, **43**, 664–675.

[14] Lauterbach, G. (1993) *Accelerating Architectural Simulation by Parallel Execution of Trace Samples*. Technical Report SMLI TR-93-22. Sun Microsystems Laboratories Inc.

[15] Wood, D. A., Hill, M. D. and Kessler, R. E. (1991) A model for estimating trace-sample miss ratios. In *Proc. 1991 SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, San Diego, CA, May 21–24, pp. 79–89. ACM, New York, NY.

[16] Nguyen, A.-T., Bose, P., Ekanadham, K., Nanda, A. and Michael, M. (1997) Accuracy and speed-up of parallel trace-driven architectural simulation. In *Proc. 11th Int. Parallel Processing Symp. (IPPS'97)*, Geneva, Switzerland, April 1–5, pp. 39–44. IEEE Computer Society, Los Alamitos, CA.

[17] Conte, T. M., Hirsch, M. A. and Hwu, W. W. (1998) Combining trace sampling with single pass methods for efficient cache simulation. *IEEE Trans. Comput.*, **47**, 714–720.

[18] Haskins, J. W. Jr. and Skadron, K. (2001) Minimal subset evaluation: Rapid warm-up for simulated hardware state. In *Proc. 2001 Int. Conf. on Computer Design (ICCD-2001)*, Austin, TX, September 23–26, pp. 32–39. IEEE Computer Society, Los Alamitos, CA.

[19] Haskins, J. W. Jr. and Skadron, K. (2003) Memory reference reuse latency: accelerated warmup for sampled microarchitecture simulation. In *Proc. 2003 IEEE Int. Symp. on Performance Analysis of Systems and Software (ISPASS-2003)*, Austin, TX, March 6–8, pp. 195–203. IEEE, Piscataway, NJ.

[20] Burger, D. C. and Austin, T. M. (1997). The SimpleScalar Tool Set. Computer Architecture News. Available at http://www.simplescalar.com.