# Self-Monitored Adaptive Cache Warm-Up for Microprocessor Simulation

Yue Luo Lizy K. John
*University of Texas at Austin, USA*
*{luo, ljohn}@ece.utexas.edu*

Lieven Eeckhout
*Ghent University, Belgium*
*leeckhou@elis.ugent.be*

## Abstract

*Simulation is the most important tool for computer architects to evaluate the performance of new computer designs. However, detailed simulation is extremely time consuming. Sampling is one of the techniques that effectively reduce simulation time. In order to achieve accurate sampling results, microarchitectural structure must be adequately warmed up before each measurement.*

*In this paper, a new technique for warming up microprocessor caches is proposed. The simulator monitors the warm-up process of the caches and decides when the caches are warmed up based on simple heuristics. In our experiments the Self-Monitored Adaptive (SMA) warm-up technique on average exhibits only 0.2% warm-up error in CPI. SMA achieves smaller warm-up error with only 1/2~1/3 of the warm-up length of previous methods. In addition, it is adaptive to the cache configuration simulated. For simulating small caches, the SMA technique can reduce the warm-up overhead by an order of magnitude compared to previous techniques. Finally, SMA gives the user some indicator of warm-up error at the end of the cycle-accurate simulation that helps the user to gauge the accuracy of the warm-up.*

## 1. Introduction

Simulation of standard benchmarks has been the most popular method for computer architects to study design tradeoffs. Modern benchmarks are no longer small kernels or synthesized toy programs. Instead, they are very close to real world programs and often take a long time to execute. Moreover, modern superscalar microprocessors are becoming increasingly complex; and so are the simulators modeling the processors. As a result, running benchmarks on detailed microarchitecture simulation models can take prohibitively large simulation times.

To reduce simulation time, several techniques have been proposed, among which sampling and reduced input sets are the most commonly used. In this paper, we focus on sampling. In a sampled simulation the original full instruction stream is divided into non-overlapping chunks of continuous instructions. Each chunk is a basic simulation unit, or a *sampling unit*. The *sampling unit size* is the number of instructions in each chunk. The *pre-sample* of a sampling unit refers to the instructions before this sampling unit up to the end of the previous sampling unit. A *sample* consists of the selected chunks that are actually simulated and measured. The number of sampling units in a sample is the *sample size*. Recently, Wunderlich et al. applied sampling theory to microarchitecture simulation [2]. Under the assumption of no measuring error, they showed that CPI can be estimated to within an error of 3% with 99.7% confidence by measuring fewer than 50 million instructions per benchmark. This accounts for only 0.029% of the average dynamic instructions executed for a benchmark program. It appears that sampling has effectively solved the problem of long simulation time.

However, the above results are obtained under the assumption that the number of cycles or CPI for each sampling unit can be accurately measured. The CPI of each sampling unit depends not only on the instructions executed in the unit, but also on the initial state of all microarchitecture structures at the beginning of this unit. The initial state is, in turn, the result of the execution of all the instructions before the sampling unit. Executing a limited number of instructions before a sampling unit to get (approximately) correct initial state is known as *warming up* the microarchitecture. The number of instructions used for warm-up before a sampling unit is its *warm-up length.* For small structures like the ROB, the reservation station, and the register file, thousands of instructions are enough to put them into correct state. However, some structures in the microprocessor like the branch target buffer, and the caches can hold thousands to millions of bytes. It is difficult to ensure that they are in the correct initial state before every sampling unit in the simulation. If the initial state is not correct, the error can be large. For example, Haskins et al reported that in their experiment, ignoring warm-up could result in an error as high as 15% in simulated CPI [12]. Thus adequate warm-up is critical to the accuracy of sampled simulation. Warm-up does not only affect accuracy but also incurs overhead and increases simulation time. When simulating a processor with large caches, a large number of instructions may be needed for adequate warm-up, which prolongs the simulation. Therefore, warm-up issue is very important in sampled simulation. A good warm-up scheme should achieve a desired level of accuracy while devoting as few instructions as possible for warm-up.

We believe that warm-up is still an important issue in sampled microprocessor simulation and deserves active research. But there is an opinion that warm-up is largely solved and little reduction in simulation time can be accomplished with better warm-up techniques. For example, MRRL is claimed to have achieved 90% of the maximum possible simulation speed [12]. However, careful analysis of the experiment reveals functional simulation as the bottleneck because every benchmark is simulated functionally from beginning to end. In our simulation environment, the relative speed of functional simulation (no microarchitecture simulation at all), functional warm-up (only cache and branch predictor simulation), and cycle-accurate simulation, is 1:1/2.8:1/16[1]. Fifty 1 million-instruction sampling units are used in [12]. Suppose that a benchmark is 100 billion instructions long and on average each sampling unit needs 30 million instructions for warm-up[2]. Then the percentages of time spent in functional simulation, functional warm-up and cycle simulation are 95.17%, 4.06%, and 0.77%. It is obvious that the functional simulation is the bottleneck, so even getting rid of warm-up overhead altogether will provide little benefit. However, if the user saves the checkpoints or the traces for each sampling unit, she no longer needs to run the benchmark from beginning to end and is able to simulate for each sampling unit directly. In this case, removing the warm-up overhead will give 6.25 times speedup in simulation! Therefore, better warm-up technique is still highly desired. (Reducing the storage cost of checkpoints/traces is among our future research.)

In this paper, we study the warm-up process of the processor caches and propose a self-monitored adaptive warm-up scheme for simulation. The simulator monitors the warm-up process of the caches and decides when the caches are warmed up based on a simple heuristic. Unlike previous research, this method is both adaptive to the characteristics of the benchmark and the cache configuration being simulated. Overall, it achieves very good accuracy with lower warm-up overhead than previously proposed techniques.

In the next section, we survey the existing warm-up techniques and discuss the strength and weakness of each technique. To overcome these weaknesses, we present our new self-monitored adaptive cache warm-up scheme in Section 3. Our proposed technique is evaluated experimentally in Section 4. Finally, we conclude and discuss future research in Section 5.

## 2. Related work

Obviously the most accurate way to warm up the caches is to do cache simulation throughout the benchmark execution. This is how the SMARTS scheme [2] does the warm-up. The simulator switches between functional warm-up and cycle-accurate simulation. During functional warm-up, the simulator executes the program without simulating the pipeline stages, but the caches and the branch predictors are simulated. During cycle simulation, the simulator does detailed cycle accurate simulation. Therefore, the only error in warm-up is introduced by not simulating the effect of out-of-order execution and wrong path execution on the caches during functional warm-up. It has been shown that this error is small [2] [3]. Although this warm-up scheme is by far the most accurate, it is still not satisfactory. First, always simulating caches can be a waste of resource. According to sampling theory, for a specific accuracy, the sample size should be determined by the variability in the population. If the benchmark does a lot of repetition, only a tiny fraction of the instruction stream is needed. However, the scheme requires that caches be simulated for every instruction, which is inefficient. Secondly, always warming up the cache makes distributed simulation hard. For sampling methods such as SimPoint [14] and Variance SimPoint [15], where a small number of relatively large sampling units are taken, each sampling unit can be simulated on different machines in parallel to greatly improve the overall simulation speed. However, constantly warming up caches as in SMARTS make it difficult to distribute the simulation on multiple machines.

Another simple warm-up scheme is to devote a fixed number of instructions to warm up the cache. This is called "PRIME" scheme [6] following Crowley et. al.'s terminology [4]. At an extreme, zero

[1] The relative speed numbers are highly dependent on the simulator and the configuration being simulated.

[2] No warm up length number is given in [12]. This number is based on our experiment with MRRL. See Section 4.2.

instructions are used to warm up the cache before cycle-simulating a sampling unit. Then two types of assumption can be made about the initial state of the cache. If all cache lines are assumed to be invalid, it is called "COLD" scheme, which, obviously, overestimates the number of cache misses. If, on the other hand, the cache state is assumed to be the same as the state at the end of last sampling unit, the scheme is called "STITCH" [5]. The efficacy of these assumptions depends on workload, cache organization, and choice of sampling parameters. The user is often left with a result whose error the user has no idea unless he/she does the full simulation. An additional problem with prime is the lack of guideline for the user to choose the number of instructions to warm up the cache.

The problem of cache warm-up is that the state of the cache is unknown at the beginning of each observation. In other words, since portions of the trace are unexamined between observations, it is unknown whether the first reference to each cache block will be a hit or a miss. Such references are referred to as *cold-start references*. Laha et al [7] proposed not counting these cold-start references when calculating cache misses. This effectively assumes that the miss rate for the cold-start references is equivalent to the miss rate for all other references. Wood et al [8] show that this assumption is usually not true. The miss rate for the cold-start references is higher than the overall miss rate. Employing a renewal theoretical model, Wood et al propose a method to estimate the miss rate for the cold-start references by observing the average live and dead time for each cache line. These two methods can be used to calculate the cache miss rate from sampled trace, but not directly applicable to microarchitectural simulation to get CPI.

Haskins and Skadron have proposed two techniques to determine the warm-up length for a sampling unit. The Minimal Subset Evaluation (MSE) [9] 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 sampling unit that must be modeled in order to achieve accurate cache state. This work only handles warm-up for the first-level data and instruction caches. In their second technique, they measure the Memory Reference Reuse Latency (MRRL) [12], which refers to the elapsed time measured in number of instructions between a reference to some memory address and the next reference to the same address. Instructions in a sampling unit and its pre-sample are profiled to get the distribution of MRRL. Given a $p$-value ($p$%) the warm-up length is the $p$-percentile of the distribution. Because most of the instructions used to calculate the distribution of MRRL are from the pre-sample, it is hard to guarantee that the instructions in the sampling unit also follow the distribution.

To avoid this problem, Eeckhout et al proposed the Boundary Line Reuse Latency (BLRL[3]) method [10], in which every memory reference in a sampling unit is directly examined instead of relying on aggregated distribution. For a reference $r$ to address $a$ generated by instruction $I$ in the sampling unit, they search for a reference $r'$ to the same address $a$ in the pre-sample. If reference $r'$ is found and it is generated by instruction $I'$, then warming up from $I'$ can guarantee that we know whether $r$ is a hit or miss when LRU replacement policy is used. Given a $p$-value like 90%, the instructions in the pre-sample of the sampling unit are scanned. The warm-up length for the sampling unit is chosen such that 90% of the unique references in the sampling unit whose addresses are referenced in the pre-sample are covered by the warm-up instructions.

Neither MRRL nor BLRL takes the cache organization into consideration. The cache warm-up process depends on both the workload and the cache organization. The methods discussed so far only consider the workload. A small direct mapped cache is intuitively easier to warm up than a large highly associative one, but these methods calls for the same warm-up length given the same $p$-value. Therefore, the cache-independent methods may be overkill depending on the cache being simulated.

## 3. Self-monitored adaptive warm-up

As discussed in the previous section, none of the existing warm-up technique is satisfactory. One major problem with MRRL and BLRL is that they do not take cache configuration into account. Different caches may require different warm-up length even for the same benchmark. Therefore, using any fixed $p$-value in the techniques may result in under-warm-up or over-warm-up for different caches. When we carefully examine the previous techniques, we see that none of them are really warm-up method per se. The warm-up method itself is simulating instructions before each sampling units. All the methods just help the user to decide when the warm-up is enough, so why not monitor the warm-up process in the simulator to decide whether the warm-up is enough? This is exactly the rationale behind the self-monitored adaptive (SMA) warm-up technique.

In SMA warm-up, as in the previous techniques, the simulator does functional warm-up before switching to

[3] The method was not given an official name in [10]. It is called "Boundary Line Reuse Latency" because it is equivalent to profiling the memory reuses that cross the boundary line between a sampling unit and its pre-sample.

detailed cycle accurate simulation. During the functional warm-up, the caches are accessed but no pipeline stages are simulated. The warm-up process of the cache is monitored. The simulator switches to cycle simulation as soon as the cache is deemed "warmed up". Therefore, the warm-up length is not fixed but adaptive. Unlike previous approaches, this technique implicitly considers both the workload characteristics and the cache organization. Fewer instructions will be used for warming up a small direct-mapped cache than for a large highly associative one.

To monitor the cache warm-up process, all the cache blocks are initialized to the *cold-start* state before the functional warm-up. The address/tag in a cold-start block is unknown because it depends on the previous instructions, which were not simulated. When a cache access is initiated, the set index to the cache can be calculated. If the memory address is not found in this set and one or more cache block in this set is in cold-start state, then we call the cache access a *cold-start access*. It is not known whether a cold-start access will result in a cache miss or a hit. When data is brought to a cold-start state cache block, the block changes to the "valid" state. Once a cache block leaves the cold-start state, it never goes back to this state again. We call any state other than the cold-start state a known state.

Two aspects of the warm-up process are monitored. Firstly, the simulator keeps track of the percentage of the cache blocks in cold-start state. This number monotonously decreases during warm-up. If no cache block is in cold-start state, the cache is completely warmed up. We can guarantee that the outcome of every future reference is known. Secondly, the simulator monitors the number of cold-start accesses during an interval. When the cache is large, or the working set of the program is small, it may take too long to completely warm up the cache. In this case, the cache is deemed warmed up when the number of cold-start accesses is below a user-defined threshold. Unlike a completely warmed up cache, there is no guarantee that all future reference will access blocks in known state. However, the possibility of cold-start accesses is low. The detailed information on choice of parameters for the interval size and threshold is given in the next section. Monitoring the warm-up process is a very low overhead operation, it only increments or decrements a couple of counters at a cold-start cache access. There is no time overhead for accessing cache blocks in known state. The number of cold-start accesses usually decreases quickly.

Another problem with the previous methods is that the user generally does not know how accurate the warm-up was after the simulation. She has to rely on previously published validated results. However, the user's configuration may not be the same as in the published paper. SMA can give the user some indication of the accuracy of the warm-up after the simulation. After switching to cycle accurate simulation, the simulator continues to count the number of cold-start accesses. In this way, after the simulation the user knows how much of all the cache misses are due to cold-start accesses. In the experiment we count a cold-start access as a cache miss. So the number of cold-start accesses is usually the upper bound of the overestimation of cache misses. For example, if during cycle accurate simulation of 1 million instructions the user only sees 20 cold-start cache references, then she knows that the overestimation of cache misses is very unlikely to go above 20 and the CPI result should be fairly accurate.

**Table 1. Benchmarks, their data set and dynamic instruction count. The data set name is appended to the benchmark name**

| Benchmark | # of Instructions (million) |
|---|---:|
| gcc-166 | 46, 918 |
| bzip2-source | 108,878 |
| crafty | 191,883 |
| eon-cook | 80,614 |
| gap | 269,036 |
| gzip-graphic | 103,706 |
| mcf | 61,867 |
| twolf | 346,485 |
| vortex-1 | 118,977 |
| vpr-route | 84,069 |

**Table 2. Processor configuration**

| Parameter | 8-way (baseline) |
|---|---|
| Machine Width | 8 |
| RUU/LSQ size | 128/64 |
| Memory System | 32KB 2-way L1 I & D, 2 ports,<br>Unified 1M 4-way L2 |
| ITLB / DTLB | 4-way 128 entries<br>4-way 256 entries<br>200 cycle miss penalty |
| L1/L2/Memory Latency | 1/12/100 cycles |
| Functional Units | 4 I-ALU<br>2 I-MUL/DIV<br>2 FP-ALU<br>1 FP-MUL/DIV |
| Branch Predictor | Combined 2K tables<br>7 cycle misprediction penalty<br>1 prediction/cycle |

## 4. Experiments and results

Ten benchmarks from SPECint 2000, listed in Table 1, are used in our experiment. The programs, downloaded from the SimpleScalar web site [16], are compiled for the Alpha ISA. All the experiments are done on our modified SimpleScalar v3.0 [1]. Table 2 shows the main processor configuration used in our experiment. This configuration is adapted from the SMARTS paper [2].

### 4.1. Variability in warm-up process

Much research has been done on devising and comparing warm-up techniques, but few of the projects shed light on the warm-up process itself. We have done experiments to study how the cache warm-up process proceeds. In this section, we only present one important issue in cache warm-up, the variability in the warm-up length. The effectiveness of the new warm-up technique depends on the variability. If a constant warm-up length is good for all situations, then the PRIME method with fixed warm-up length will be the best. However, if the warm-up length changes widely, then a good warm-up technique needs to adapt to all the factors that affect the warm-up process.

Each benchmark execution is divided into *segments* of 100 million instructions. We study the cache warm-up process of each segment, so the simulator sets all cache blocks to cold-start state at the beginning of each segment. We track the warm-up process in each segment. For L1 data cache, we record, for each segment, the warm-up length needed to put every cache block in known state. Table 3 lists the average and the standard deviation of warm-up length. The L2 cache may not completely warm up at the end of 100 million instruction segments, so we record the warm-up length needed to warm up 50% of the cache blocks for each segment. The statistics for the L2 cache warm-up length is shown in Table 4. These warm-up lengths are not the warm-up length required in simulation. Nevertheless they reflect the large variability in the warm-up process. As we can see, the warm-up length is different for different benchmarks. It is also widely different within one benchmark. The standard deviation of the warm-up length of different segments is as large as the mean in many cases. Therefore, devoting fixed number of instructions to warm-up as in PRIME method is not good. Comparing Table 3 and Table 4 we also see that the large L2 cache needs much longer warm-up than the small L1 cache. Therefore, it is important for a good warm-up method to also take into consideration of the cache configuration. MRRL and BLRL both adapt to the different segment in a benchmark but they cannot adapt to the cache configuration.

**Table 3. Warm-up length for warming up all cache blocks in L1 data cache (in 100,000 instructions).**

| Benchmark | Average | Standard Deviation | Max | Min |
|---|---|---|---|---|
| bzip2-source | 17.80 | 16.82 | 184 | 1 |
| gcc-166 | 9.98 | 15.92 | 145 | 1 |
| crafty | 110.61 | 51.59 | 439 | 21 |
| eon-cook | 27.87 | 14.36 | 106 | 7 |
| gap | 8.08 | 10.28 | 167 | 1 |
| gzip-graphic | 4.62 | 2.88 | 14 | 1 |
| mcf | 1.54 | 4.02 | 45 | 1 |
| twolf | 2.82 | 15.04 | 687 | 2 |
| vortex-1 | 14.46 | 15.01 | 141 | 1 |
| vpr-route | 3.59 | 3.94 | 66 | 1 |

**Table 4. Warm-up length for warming up 50% cache blocks in L2 cache (in 100,000 instructions).**

| Benchmark | Average | Standard Deviation | Max | Min |
|---|---|---|---|---|
| bzip2-source | 177.32 | 233.17 | 999 | 1 |
| gcc-166 | 546.30 | 331.57 | 999 | 1 |
| crafty | 303.68 | 165.11 | 986 | 28 |
| eon-cook | 155.47 | 272.07 | 998 | 2 |
| gap | 136.75 | 62.20 | 788 | 3 |
| gzip-graphic | 837.82 | 245.56 | 999 | 5 |
| mcf | 15.41 | 73.19 | 810 | 1 |
| twolf | 34.76 | 22.38 | 920 | 8 |
| vortex-1 | 208.94 | 84.66 | 874 | 5 |
| vpr-route | 52.69 | 35.47 | 232 | 2 |

### 4.2 Comparison with MRRL and BLRL

In this section we compare SMA with the two most recently proposed warm-up techniques, MRRL and BLRL. We compare both the warm-up length and the accuracy in CPI. In the experiment, we choose a sampling unit size of 1 million instructions. This sampling unit size was used in MRRL paper [12], and Variance SimPoint [15]. In this section, each benchmark execution is divided into segments of 200 million instructions. We use 200 million instruction segment size instead of 100 million in the previous section to give larger gap between sampling units for more accurate profiling in MRRL and BLRL. One sampling unit is chosen from each segment.

In SMA the sampling units are not previously determined but rather depend on the cache warm-up process. Once the cache is deemed warmed up enough, the simulator executes 4,000 instructions in cycle accurate mode to warm up the pipeline as suggested in [2], and then the CPI of 1 million instruction sampling unit is measured. As discussed in the above section, the L2 cache may not be completely warmed up with reasonable number of instructions so we cannot use complete warm-up as the only criterion for the caches. Therefore, we choose the following simple heuristic to judge if the cache is warmed up. At the end of each interval, we calculate the average number of cold-start accesses for the last *N* intervals. If the average number of cold-start references falls below a threshold *T*, we assume that the cache is warmed up enough and end the functional warm-up. Because this method requires warm-up of at least *N* intervals, to take advantage of segments that reach complete warm-up quickly, we also monitor the number of cache blocks in the cold-start state in the cache. The functional warm-up also ends as soon as the cold-start state blocks drop to zero. For L1 data cache, we use *N*=20, *T*=10. For L1 instruction cache, we use *N*=10, *T*=1. For L2 cache, we use *N*=20, *T*=15.

For MRRL and BLRL the sampling units are chosen to be the same as those in SMA. 4,000 instructions are also simulated in the cycle-accurate mode before each sampling units to warm up the pipeline. The profiler for MRRL was downloaded from its author's website [18].

Although not implemented in the current simulator, we hope to further improve simulation speed by distributed simulation. When sampling units are distributed to different machines, the end state of one sampling unit cannot be used as the beginning state of another sampling unit. Therefore, in our experiment caches are emptied before warming up each sampling unit.

The final error in CPI in sampled simulation comes from two sources: the sampling error per se and the warm-up error. To fairly compare different warm-up techniques, only the warm-up error should be measured, so we additionally run a simulation with full cache warm-up. In this simulation the caches are always simulated between every sampling units as in SMARTS [2]. The sampling units and the cycle-accurate warm-up are the same in all of our simulations, so the difference between the CPI of a warm-up technique and the CPI of full cache warm-up is the warm-up error.

Table 5 compares SMA with MRRL and BLRL. The heuristic in SMA relies on the warm-up history to predict whether the cache is warmed up enough in the next sampling unit, so SMA may mispredict and end functional warm-up prematurely. In Table 5 the average error is only about 0.2%, so SMA is very accurate and rarely mispredicts.

For MRRL, we choose the *p*-value to be 99.9%, which is the default value suggested in [12]. For BLRL, we use the *p*-value of 90%. Both methods are also accurate, exhibiting an average error of 0.4% and 0.3%. However, SMA clearly shows the overall advantage. The SMA technique requires only 1/3 of the warm-up length of MRRL or 1/2 of the warm-up length of BLRL yet it achieves an error that is smaller than the other two techniques.

SMA is better in both warm-up length and accuracy, so changing the *p*-value for all benchmarks will not affect the overall conclusion. Using different *p*-values for different benchmarks may improve the overall result of MRRL and BLRL, but asking the user to fine tune the *p*-value for each benchmark and different processor configuration is not practical.

## 4.3. Adaptivity to cache configuration

Unlike previous methods, SMA adapts to the cache configuration being simulated. In the last section, we show how SMA performs with a cache size that is common to workstations. To evaluate its adaptivity, in this section we simulate a small cache configuration that is typical in an embedded processor. Table 6 shows the cache configuration used in our experiment,

**Table 5. Comparison of SMA with MRRL and BLRL**

| **Benchmark** | **# of sampling units** | **Avg warm up length per sampling unit (1000 instructions)** | | | **Error in CPI** | | |
|---|---|---|---|---|---|---|---|
| | | **SMA** | **MRRL** | **BLRL** | **SMA** | **MRRL** | **BLRL** |
| bzip2-source | 545 | 8,710 | 100,227 | 78,577 | 0.1440% | 0.0124% | 0.0546% |
| gcc-166 | 235 | 13,221 | 7,064 | 12,369 | 0.1860% | 0.9730% | 0.4667% |
| crafty | 960 | 5,337 | 3,688 | 14,868 | 1.0300% | 1.5700% | 1.0374% |
| eon-cook | 403 | 14,367 | 6,667 | 3,191 | 0.0419% | 0.2974% | 0.2805% |
| gap | 1346 | 8,861 | 11,629 | 12,094 | 0.0434% | 0.9620% | 0.1123% |
| gzip-graphic | 519 | 9,315 | 7,945 | 5,227 | 0.5110% | 0.1710% | 0.0865% |
| mcf | 310 | 2,706 | 34,739 | 7,611 | 0.0039% | 0.0177% | 0.0254% |
| twolf | 1733 | 5,970 | 15,308 | 5,420 | 0.2070% | 0.0207% | 0.2606% |
| vortex-1 | 595 | 23,162 | 39,095 | 34,878 | 0.1400% | 0.1918% | 1.0884% |
| vpr-route | 421 | 7,624 | 91,877 | 55,362 | 0.0269% | 0.0066% | 0.0236% |
| **Average** | | **9,927** | **31,824** | **22,960** | **0.2334%** | **0.4223%** | **0.3436%** |

which is modeled after Intel XScale PXA255 embedded processor. Although SPECint is not the best benchmark suite for embedded processor, we still choose it so that we can compare with the warm-up length for the large cache configuration. Because we do not need to profile for MRRL or BLRL in this experiment we use a segment size of 100 million instructions to increase sample size. Using the same warm-up heuristic parameters as in the previous section, the average warm-up length per sampling unit for different benchmarks is shown in Figure 1. The first bar for each benchmark shows the warm-up length for the large cache configuration in Table 2, and the second bar is the warm-up length for the small cache configuration in Table 6. It is clear that SMA adapts well to the cache configuration. For the small caches the warm-up length is on average only 1/6 of that required by the large caches.

**Table 6. Configuration for small caches**

| Cache | Block Size (bytes) | Associ-ativity | # of Sets | Replacement Policy |
|---|---|---|---|---|
| L1 Data | 32 | 32 | 32 | LRU |
| L1 Instr | 32 | 32 | 32 | LRU |
| L2 | None | | | |

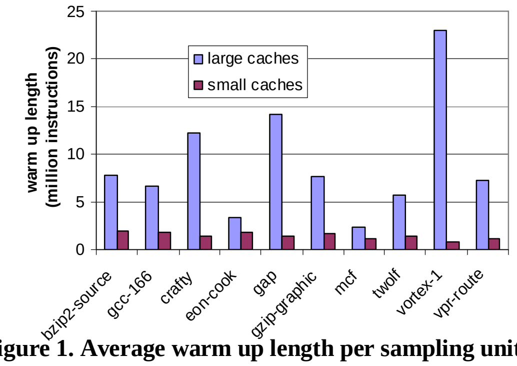

**Figure 1. Average warm up length per sampling unit for different cache configurations**

Neither MRRL nor BLRL can adapt to the cache configuration. Using the warm-up length for MRRL or BLRL in the previous section for the small caches will result in 15~20X larger overhead than SMA. The only way to reduce warm-up length for the two techniques is to reduce the *p*-value. However, to come up with a good *p*-value for each configuration by experiment is highly impractical and defeats the goal of reducing simulation time.

SMA reduces warm-up overhead for small caches, but we also need to make sure that it does not compromise the accuracy of warm-up. We do not have a good microarchitecture configuration for Intel XScale PXA255 so we did not do the cycle accurate simulation. Instead, we do the cache simulation and measure the accuracy in cache misses. Inadequate warm-up will cause overestimation of cache misses and eventually lead to error in CPI. Table 7 shows the absolute error in the number of data cache misses per sampling units compared with full cache warm-up. This error is extremely small so SMA does not lose accuracy when adapting to the small caches.

**Table 7. Absolute error in the number of data cache misses per sampling unit.**

| Benchmark | Error |
|---|---|
| bzip2-source | 0 |
| gcc-166 | 0 |
| crafty | 0.002614 |
| eon-cook | 0 |
| gap | 0 |
| gzip-graphic | 0 |
| mcf | 0 |
| twolf | 0.00318 |
| vortex-1 | 0 |
| vpr-route | 0 |

## 5. Conclusion and discussion

Sampling can greatly reduce simulation time. However, effective sampling requires efficient and accurate warm-up of microarchitecture structures. In this paper, we studied the warm-up process of microprocessor caches. It is found that the warm-up process varies widely for different benchmarks, for different portion in one benchmark execution, and for different cache configurations. Based on this observation, we propose the self-monitored adaptive cache warm-up scheme. The simulator monitors the cache warm-up process and decides when the warm-up is enough based on a simple heuristic. The experiments show that SMA is accurate, exhibiting an average warm-up error of about 0.2%. SMA does not only offers superior overall accuracy but also reduces the warm-up length to 1/2 ~ 1/3 of two recently proposed methods. Unlike previous methods, SMA is adaptive to cache configuration so it can reduce warm-up overhead by an order of magnitude for simulating small caches. Because SMA continues to monitor the cache accesses during cycle accurate simulation, the user can get the number of cold-start cache accesses in each sampling unit as an indicator of the accuracy of the warm-up.

SMA has one weakness: the user does not know beforehand when the cycle-accurate simulation begins. This is not a problem for simple random sampling, but poses difficulty for sampling techniques that use a predefined set of sampling units such as SimPoint. Currently we are looking into statistical sampling theory for designing better sampling techniques to work with SMA.

SMA also looks promising for warming up other microarchitecture structures such as the branch predictor and the value predictor. Both of them share the same property with caches that once an element is warmed up, it never goes back to cold-start state again, so they are also candidate for SMA. Unlike caches, one access to a branch table element is not sufficient to put it into a known state, so designing accurate warm-up method by tracking reuse latency as in MRRL or BLRL is not easy, but monitoring the warm-up process with Vengroff et al's deterministic finite automaton [17] may be much simpler. Extending SMA to warm up other structures is also part of our future research.

## Acknowledgements

This research is 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. Lieven Eeckhout is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium) (F.W.O. Vlaanderen).

## References

[1] D. Burger, and T. M. Austin, The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin-Madson, June 1997.

[2] R. E.Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In Proceedings of the 30th Annual International Symposium on Computer Architecture, pp 84-95. June 2003.

[3] H. W. Cain, K. M. Lepak, B. A. Schwartz, and M. H. Lipasti, "Precise and accurate processor simulation", 5th Workshop On Computer Architecture Evaluation Using Commercial Workloads (CAECW), February 2002.

[4] P. Crowley and J. L. Baer. On the Use of Trace Sampling for Architectural Studies of Desktop Applications. Proceedings of the 1999 SIGMETRICS Conference. May 1999.

[5] A. Agarwal, M. Horowitz, and J. Hennessy. An analytical cache model. ACM Transactions on Computer Systems, 7(2):184–215, May 1989.

[6] J. W. C. Fu, and J. H. Patel. Trace driven simulation using sampled traces. In Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences Vol. I: Architecture, pp 211–220, January 1994.

[7] S. Laha, J. H. Patel, and R. K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. IEEE Transactions on Computers, 37(11):1325–1335, November 1988.

[8] D. A. Wood, M. D. Hill, and R. E. Kessler. A model for estimating trace-sample miss ratios. In Proceedings of the ACM SIGMETRICS Conference for the Measurement and Modeling of Computer Systems, pp 79–89, June 1991.

[9] 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.

[10] L. Eeckhout, S. Eyerman, B. Callens, K. De Bosschere. Accurately warmed-up trace samples for the evaluation of cache memories. High Performance Computing Symposium 2003, Orlando, Florida.

[11] E. Perelman, G. Hamerly, and B. Calder, Picking Statistically Valid and Early Simulation Points, International Conference on Parallel Architectures and Compilation Techniques, September 2003

[12] J. W. Haskins, Jr. and K. Skadron. Memory Reference Reuse Latency: Accelerated Sampled Microarchitecture Simulation. In Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software, pp. 195-203, Mar. 2003

[13] Z. Hu, S. Kaxiras and M. Martonosi. Timekeeping in the Memory System: Predicting and Optimizing Memory Behavior. The 29th International Symposium on Computer Architecture (ISCA-29), May 2002.

[14] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems pp45-57, October 2002.

[15] Erez Perelman, Greg Hamerly, and Brad Calder, Picking Statistically Valid and Early Simulation Points , International Conference on Parallel Architectures and Compilation Techniques, September 2003.

[16] SimpleScalar LLC. http://www.simplescalar.com/

[17] Darren Vengroff and Guang Gao. Partial sampling with reverse state reconstruction: A new technique for branch predictor performance estimation. Fourth International Symposium On High-Performance Computer Architecture (HPCA), 1998.

[18] J. W. Harskins. Memory Reference Reuse Latency: Rapid Warm Up for Sampled Microarchitecture Simulation. http://www.cs.virginia.edu/~jwh6q/mrrl-web/