Branch History Matching: Branch Predictor Warmup for Sampled Simulation

Simon Kluyskens Lieven Eeckhout

ELIS Department, Ghent University
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Email: leeckhou@elis.UGent.be

Abstract

Computer architects and designers rely heavily on simulation. The downside of simulation is that it is very time-consuming — simulating an industry-standard benchmark on today's fastest machines and simulators takes several weeks. A practical solution to the simulation problem is sampling. Sampled simulation selects a number of sampling units out of a complete program execution and only simulates those sampling units in detail. An important problem with sampling however is the microarchitecture state at the beginning of each sampling unit. Large hardware structures such as caches and branch predictors suffer most from unknown hardware state. Although a great body of work exists on cache state warmup, very little work has been done on branch predictor warmup.

This paper proposes Branch History Matching (BHM) for accurate branch predictor warmup during sampled simulation. The idea is to build a distribution for each sampling unit of how far one needs to go in the pre-sampling unit in order to find the same static branch with a similar global and local history as the branch instance appearing in the sampling unit. Those distributions are then used to determine where to start the warmup phase for each sampling unit for a given total warmup length budget. Using SPEC CPU2000 integer benchmarks, we show that BHM is substantially more efficient than fixed-length warmup in terms of warmup length for the same accuracy. Or reverse, BHM is substantially more accurate than fixed-length warmup for the same warmup budget.

1 Introduction

Architectural simulations are extensively used by computer architects and designers for evaluating various design tradeoffs. Unfortunately, architectural simulation is very time-consuming. Even on today's fastest machines and simulators, simulating an industry-standard benchmark easily takes several weeks to run to completion. As such, simulating entire benchmark executions is infeasible for exploring huge microarchitecture design spaces. Therefore, researchers have proposed sampled simulation [1,2,3,4]. Sampled simulation takes a number of so called sampling units that are simulated in detail. Statistics or appropriate weighting is then applied to the simulation results of the various sampling units for predicting the performance of the overall benchmark execution.

An important issue with sampled simulation is the microarchitecture state at the beginning of each sampling unit, *i.e.*, the microarchitecture state at the beginning of a

sampling unit is unknown during sampled simulation. This is well known in the literature as the cold-start problem. A solution to the cold-start problem is to warmup various microarchitecture structures prior to each sampling unit. A large amount of work has been done on cache structure warmup. However, the amount of work done on branch predictor warmup is very limited.

This paper proposes *Branch History Matching (BHM)* as a novel branch predictor warmup method. The basic idea is to inspect the pre-sampling unit, *i.e.*, the instructions in the dynamic instruction stream prior to the sampling unit, for branch instances of the same static branch with similar global and local histories as the branch instances in the sampling unit. A BHM distribution is then built for all sampling units that quantifies the locality in the branch execution stream taking into account both the global and local histories of the branches. As a final step, the appropriate warmup length is then determined for each sampling unit taking into account the BHM distributions as well as the total warmup budget. In other words, the total warmup budget is distributed across the various sampling units according to the BHM distribution. Sampling units that show good locality are given a small warmup length; sampling units that show poor locality are given a larger warmup length.

BHM is microarchitecture-independent, *i.e.*, the warmup lengths are computed once and are then reused across branch predictors during design space exploration. An appealing way of using BHM in practice for sampled processor simulation is to use (i) checkpointed sampling [5,6] maintaining reduced checkpoints of architecture state (registers and memory) along with (ii) checkpointed cache warmup [5,7,8,9] and (iii) compressed branch traces [10] that are reduced through BHM. In other words, instead of having branch traces of full benchmark executions as proposed in [10], BHM limits the length of the compressed branch traces. This would result in a reduction in required disk space as well as a reduction in overall simulation time while pertaining the advantage of compressed branch traces of being branch predictor independent.

This paper makes the following contributions:

- First, we show that branch predictor warmup is an issue when it comes to guaranteeing an accurate hardware state at the beginning of a sampling unit. We show that for small sampling unit sizes, branch predictor warmup is required in order to achieve an accurate estimate of the hardware state at the beginning of the sampling unit. We provide results showing that even for (fairly large) 1M instruction sampling units branch predictor warmup is required.
- Second, we propose Branch History Matching (BHM) as a novel branch predictor warmup approach. Using the SPEC CPU2000 integer benchmarks and 10K-instruction sampling units, we show that BHM is 39% more accurate than fixed-length warmup for the same warmup length. Or reverse, BHM achieves the same accuracy as fixed-length warmup with a 1.6X shorter warmup length. Compared to MRRL, BHM is 87% more accurate.

This paper is organized as follows. We first revisit sampled simulation and cover the main issues related to sampled simulation. We then present BHM as a branch predictor warmup method. We subsequently evaluate BHM and compare it against fixed-length warmup and MRRL. And finally, we conclude.

2 Sampled simulation background

In sampled simulation, a number of sampling units are chosen from a complete benchmark execution. Those sampling units are then simulated in detail; the pre-sampling units, *i.e.*, the instructions prior to a given sampling unit, are skipped. The performance of the complete benchmark is then estimated by simply aggregating or weighting the performance numbers from the various sampling units.

There are basically three issues with sampled simulation. First, the sampling units need to be chosen in such a way that the sampling units are representative for the entire program execution. Various authors have proposed various approaches for achieving this, such as random sampling [1], periodic sampling as done in SMARTS [3] and targeted sampling based on program phase behavior as done in SimPoint [2].

The second issue is how to get to those sampling units. In other words, the architecture state (register and memory state) needs to be reconstructed so that all sampling units can be functionally simulated in a correct way. This can be achieved through fast-forwarding or (reduced) checkpointing [5,9]. Checkpointing is especially beneficial for the parallel simulation of sampling units [11,12].

The third issue with sampled simulation is to estimate the microarchitecture state at the beginning of each sampling units. The microarchitecture structures that suffer the most from the cold-start problem are cache structures and branch predictors. We will discuss warmup approaches tailored towards these types of hardware structures in the following two subsections.

2.1 Cache warmup

Given the fact that caches have the largest state in a microprocessor, they are likely to suffer the most from inaccurate microarchitecture warmup. In fact, most of the prior research on the cold-start problem has been done on cache warmup. Various approaches have been proposed such as no warmup, stale state (also called stitch) [13], fixed warmup [1], cache miss rate estimators [14], no-state-loss [12,15], minimal subset evaluation (MSE) [16], memory reference reuse latency (MRRL) [17], boundary line reuse latency (BLRL) [8,18], self-monitored adaptive cache warmup (SMA) [19], memory hierarchy state (MHS) [5], memory timestamp record (MRT) [7], *etc*.

2.2 Branch predictor warmup

Compared to the amount of work done on cache warmup, very little work has been done on branch predictor warmup.

The first paper dealing with branch predictor warmup was by Conte *et al.* [1]. They proposed two approaches to branch predictor warmup, namely stale state and fixed-length warmup. Stale state (or stitch) means that the branch predictor state at the end of the previous sampling unit serves as an approximation for the branch predictor state at the beginning of the current sampling unit. An important disadvantage of stale state is that it serializes the simulation of the various sampling units, *i.e.*, it is impossible to simulate the current sampling unit without having finalized the simulation of the

previous sampling unit. Fixed-length warmup is a simple-to-implement method that achieves good accuracy if sufficiently long warmup lengths are chosen.

The second paper mentioning branch predictor warmup is by Haskins and Conte [17,20] in which they propose memory reference reuse latency (MRRL). The idea of MRRL is to look in the pre-sampling unit how far one needs to go in order to encounter the same static branch as the one in the sampling unit. MRRL computes the reuse latency, *i.e.*, the number of instructions between the branch instance in the pre-sampling unit and the one in the sampling unit, for all branch instances in the pre-sampling unit and sampling unit. For a given target cumulative probability, for example 99.5%, it is then determined where warmup should start in the pre-sampling unit. During this warmup period, the branch predictor is warmed up but no misprediction rates are computed.

A number of papers have proposed checkpointed sampling techniques [5,7,9] in which the architecture state is stored on disk, as mentioned above. These techniques typically use checkpointed microarchitecture warming for warming cache state, such as memory timestamp record [7], live-points [9] and memory hierarchy state (MHS) [5]. They suggest to store the branch predictor state as part of the microarchitecture state for the various branch predictors one may be interested in during design space exploration. This can be space-inefficient in case multiple branch predictors need to be stored, and in addition, it prevents from simulating a branch predictor that is not contained in the microarchitecture warmup.

For addressing this problem, Barr and Asanovic [10] propose to employ branch trace compression. They store a compressed branch trace on disk and upon branch predictor warming they simply decompress the compressed branch trace and use the decompressed trace for branch predictor warming. This approach is branch predictor independent and can be used to warm any branch predictor during sampled simulation. The branch trace compression scheme by Barr and Asanovic [10] however does not address the issue of how far one needs to go back in the pre-sampling unit. They assume that the entire branch trace from the beginning of the benchmark execution up to the current sampling unit needs to be compressed and decompressed. This can be time-consuming in practice, especially for sampling units deep down the benchmark execution. BHM as proposed in this paper can be used to cut down the branch traces that need to be compressed. This saves both disk space and simulation time, while keeping the benefit of the warmup approach to be branch predictor independent.

3 The need for branch predictor warmup

Branch predictors need to be warmed up during sampled simulation. This is illustrated in Figure 1 where the number of branch mispredictions per thousand instructions (MPKI) is shown for `gcc` for four sampling unit sizes: 10K, 100K, 1M and 10M instruction sampling unit sizes. Note this is in the range of sampling units used in contemporary sampled simulation environments such as SMARTS [3,9] (sampling unit size of 10K instructions) and SimPoint [2,5,21] (sampling unit sizes from 1M to 100M instructions). Each graph shows the MPKI for four (fairly aggressive) branch predictors: a 128Kbit gshare predictor, a 256Kbit local predictor, a 128Kbit bimodal predictor and a 192Kbit hybrid predictor — more details about the experimental setup and the branch

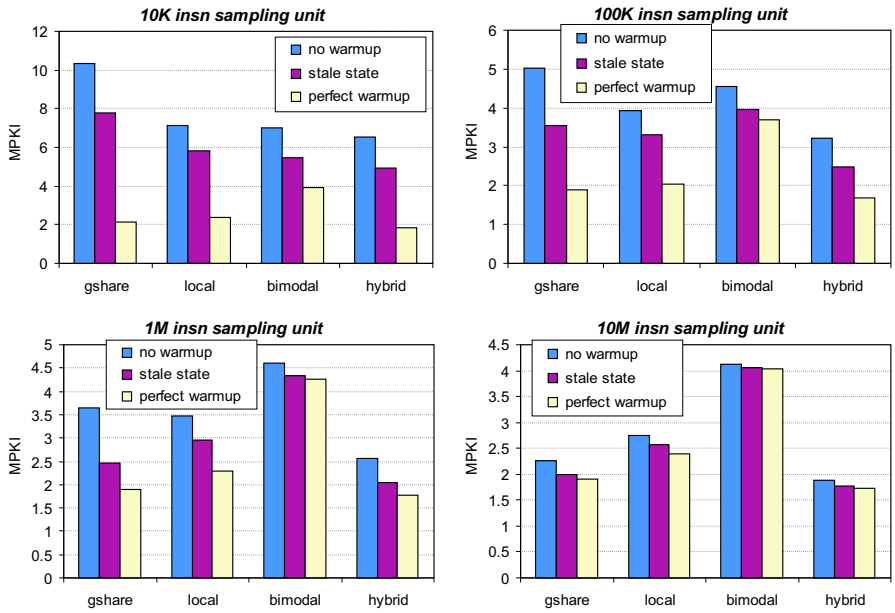

Fig. 1. No warmup, stale state and perfect warmup MPKI results for gcc and 4 branch predictors and 4 sampling unit sizes.

predictors are given in section 5. The various bars correspond to various branch predictor warmup strategies: no warmup, stale state and perfect warmup. The no warmup approach assumes an initialized branch predictor at the beginning of a sampling unit, *i.e.*, the branch predictor content is flushed at the beginning of the sampling unit — two-bit saturating counters in adjacent entries are initialized in alternate '01' '10' states. The stale state approach assumes that the branch predictor at the beginning of the sampling unit equals the branch predictor state at the end of the previous sampling unit. Note that the stale state approach assumes that sampling units are simulated sequentially — this excludes parallel sampled simulation. The perfect warmup approach is an idealized warmup scenario where the branch predictor is perfectly warmed up, *i.e.*, the branch predictor state at the beginning of the sampling unit is the state as if all instructions prior to the sampling unit were simulated.

Figure 1 clearly shows that the no warmup and stale state approaches fail in being accurate, especially for small sampling unit sizes. For example for 10K instruction sampling units, the $\Delta MPKI$ can be very high for both no warmup and stale state. Even for 1M instruction sampling units, the error can be significant, more than 1.5 $\Delta MPKI$ for the no warmup strategy and the gshare predictor. Note that the error varies across branch predictors. The error is typically higher for the gshare predictor than for the bimodal predictor, which is to be understood intuitively, the reason being the fact that the XOR hashing in the gshare predictor typically results in more entries being accessed in the branch predictor table than the bimodal predictor does.

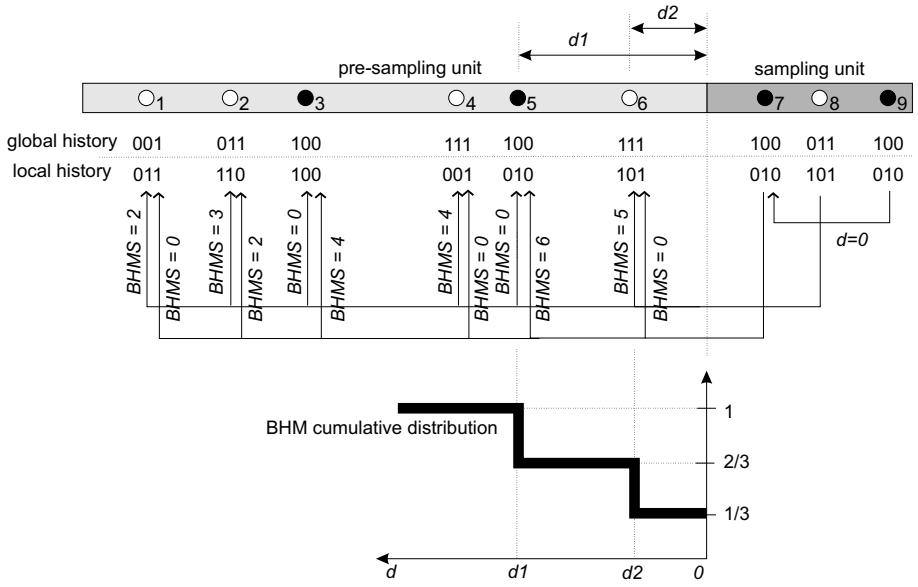

Fig. 2. An example illustrating how the cumulative Branch History Matching distribution is computed.

As a result of the non-uniform warmup error across branch predictors, incorrect design decisions may be taken. For example, using the no warmup approach, a computer architect would conclude that the local predictor achieves a better accuracy (a lower MPKI) than the gshare predictor. This is the case for 10K, 100K and even 1M instruction sampling units. However, this conclusion is just an artifact of the inadequate warmup approach. Perfect warmup shows that the gshare predictor outperforms the local predictor. The stale state warmup approach only solves this problem for the 1M instruction sampling unit, however, it does not solve the problem for smaller sampling unit sizes and it cannot be used for parallel sampled simulation.

4 Branch History Matching

This paper proposes Branch History Matching (BHM) as a novel branch predictor warmup approach. Computing the branch predictor warmup length through Branch History Matching (BHM) is done in two steps. First, we compute the BHM distribution for all sampling units. In a second phase, we then determine the warmup length for each sampling unit for a given total warmup length budget using the BHM distributions for all sampling units.

4.1 Computing the BHM distribution

Computing the BHM distribution for a given sampling unit is illustrated in Figure 2. At the top of Figure 2, a sampling unit along with its pre-sampling unit is shown. The bullets represent a single static branch being executed multiple times in the pre-sampling

unit as well as in the sampling unit. Instructions with labels '1' thru '6' are part of the pre-sampling unit; instructions labeled '7', '8' and '9' are part of the sampling unit. A white bullet represents a non-taken branch; a black bullet shows a taken branch. Figure 2 also shows the global and local history for each dynamic instance of the given static branch; the example assumes three global history bits and three local history bits. Note that the most recent branch outcome is shifted in on the right hand side of the history register; for example, a non-taken branch changes the local history from '011' to '110'.

In order to compute the BHM distribution, we first compute the BHM histogram. The BHM histogram is computed by scanning all the branch instances in the sampling unit and proceeds as follows.

- *Searching the sampling unit.* We first determine whether there is a perfect match for the local and global history of the given branch instance in the sampling unit versus the local and global histories of all the preceding branch instances of the same static branch in the sampling unit. A perfect match means that both the local and global histories are identical for the two respective branch instances. For the example given in Figure 2, the local and global histories of branch instance '9' in the sampling unit show a perfect match with the local and global history of branch instance '7' in the sampling unit. This case increments the count for $d = 0$ in the BHM histogram.
- *Searching the pre-sampling unit.* In case there is no perfect match with a preceding branch instance in the sampling unit, we search the pre-sampling unit for the most recent branch instance that shows the highest match with the local and global history for the given branch instance. This is done by computing the Branch History Matching Score (BHMS) between the given branch instance in the sampling unit with all the branch instances of the same static branch in the pre-sampling unit. The BHMS between two branch instances is computed as the number of bit positions that are identical between the local and global histories of the respective branch instances. When computing the number of identical bit positions we count from the most recent bit to the least recent bit and we stop counting as soon as there is disagreement for a given bit, *i.e.*, we count the matching most recent history bits. This done for both the global and local histories; the overall BHMS then is the sum of the global and local BHMSs. Computed BHMSs are shown in Figure 2 for the first and second branch instances of the sampling unit. For example, the BHMS for branch instance '8' with relation to branch instance '4' equals 4, *i.e.*, 2 (compare global histories '0$\underline{11}$' versus '1$\underline{11}$') plus 2 (compare local histories '1$\underline{01}$' versus '0$\underline{01}$').

  The first branch instance (with label '7') achieves a perfect match (BHMS equals 6) for the branch instance with label '5'. The idea is then to update the BHM histogram reflecting the fact that in order to have an accurate warmup for instruction '7' we need to go back to instruction '5' in the pre-sampling unit. For this purpose, the BHM histogram is incremented at distance $d1$ with 'd1' being the number of instructions between the branch instance with label '5' and the beginning of the sampling unit — this is to say that branch predictor warmup should start at branch instruction '5'. For the second branch instance (with label '8') in the sampling unit,

```
/* this function computes the current warmup length */
int current_warmup_length (int* d) {
  for (i = 0; i < n; i++)
    sum += d[i];
  return sum;
}

/* main algorithm */

/* initialize warmup length for each sampling unit */
for (i = 0; i < n; i++)
  d[i] = 0;


/* iterate as long as the user defined total warmup length L_w is not reached */
while (current_warmup_length (d) < L_w) {

  /* find the sampling unit max_j that faces the maximum slope */
  max_prob = 0.0;
  max_i = -1;
  for (i = 0; i < n; i++) {
    if ((P[i][d[i] + b] - P[i][d[i]])/b > max_prob) {
      max_prob = (P[i][d[i] + b] - P[i][d[i]])/b;
      max_i = i;
    }
  }

  /* update warmup length for sampling unit facing the maximum slope */
  d[max_i] += d[max_i] + b;

}
```

Fig. 3. The algorithm in pseudocode for determining the warmup length per sampling unit using BHM distributions.

the highest BHMS is obtained for the branch instance with label '6'; the number of instructions between that branch instance and the sampling unit starting point is denoted as $d2$ in Figure 2. We then increment the BHM histogram at distance $d2$.

Dividing the BHM histogram with the number of branch instances in the sampling unit, we then obtain the BHM distribution. Figure 2 shows the cumulative BHM distribution for the given sampling unit: since there are three branch instances in our example sampling unit, the cumulative distribution starts at $1/3$ for distance $d = 0$, reaches $2/3$ at distance $d = d2$ and finally reaches 1 at distance $d = d1$.

4.2 Determining warmup length

Once the BHM distribution is computed for each sampling unit we determine the warmup length per sampling unit for a given total warmup length budget. The goal is to partition a given warmup length budget over a number of sampling units so that accuracy is maximized. In other words, sampling units that do not require much warmup, are granted a small warmup length; sampling units that require much more warmup are given a much larger warmup length.

The algorithm for determining the appropriate warmup length per sampling unit works as follows, see also Figure 3 for the pseudocode of the algorithm. We start from n BHM distributions, with n being the number of sampling units. In each iteration, we determine the sampling unit i out of the n sampling units that faces the maximum

predictor	configuration
gshare	16-bit history gshare predictor, 128Kbit total state
local	16-bit local predictor, 8K entries at first level, 64K entries at second level 256 Kbit total state
bimodal	64K-entry bimodal predictor, 128Kbit total state
hybrid	hybrid predictor consisting of a 32K-entry bimodal predictor, a 15-bit history gshare predictor and a 32K-entry PC-indexed meta predictor; 192Kbit total state

Table 1. The branch predictors considered in this paper.

slope in the BHM distribution. This means that the sampling unit i (called `max_i` in the pseudocode in Figure 3) is determined that maximizes the slope $\frac{P_i(d_i+b)-P_i(d_i)}{b}$, with $P_i(d)$ being the probability for distance d in the cumulative BHM distribution for sampling unit i, and d_i being the warmup length granted to sampling unit i in the current state of the algorithm. For the sampling unit i that maximizes the slope, we increase the granted warmup length d_i to $d_i + b$. This algorithm is iterated until the total warmup length over all sampling units equals a user-defined maximum warmup length L_w, *i.e.*, $\sum_{i=1}^{n} d_i = L_w$. By doing so, we effectively budget warmup to samples that benefit the most from the granted warmup.

Note that this algorithm is only one possible design point in BHM warmup. More in particular, this algorithm heuristically determines to increase the warmup length for the sampling unit that faces the maximum slope in the BHM distribution. The algorithm does not take into account the distance over which this slope is observed; taking this distance into account for determining appropriate warmup lengths would be an interesting avenue for future work though.

4.3 Discussion

Most branch predictors in the literature as well as in today's commercial processors use a global and/or local branch history. Because BHM is also based on global and local branch history matching, it is to be expected that BHM will be an appropriate warmup technique for most branch predictors considered today. However, some branch predictors proposed in the literature are path-based and use a sequence of recent branch addresses as the branch history. In this paper though, we limit ourselves to branch predictors that are based on global and local branch histories. However, as part of our future work, we will further evaluate BHM for a broader range of branch predictors than the ones used in this paper.

5 Experimental setup

We use SPEC CPU2000 integer benchmarks with reference inputs in our experimental setup. We include all integer benchmarks except for perlbmk because its branch misprediction rate is very low; in fact, no warmup is very accurate for perlbmk. The binaries which were compiled and optimized for the Alpha 21264 processor, were taken from the SimpleScalar website. All measurements presented in this paper are obtained using the binary instrumentation tool ATOM [22]. The branch predictors considered in

this paper are shown in Table 1. We consider four fairly aggressive branch predictors: a gshare predictor, a local predictor, a bimodal predictor and a hybrid predictor [23,24].

Our primary metric for quantifying the accuracy of the branch predictor warmup approaches proposed in this paper is $\Delta MPKI$ which is defined as the absolute difference between the number of misses per thousand instructions under perfect warmup ($MPKI_{perfect}$) versus the number of misses per thousand instructions under the given branch predictor warmup approach ($MPKI_{warmup}$). In other words, $\Delta MPKI = \|MPKI_{warmup} - MPKI_{perfect}\|$ and thus the smaller $\Delta MPKI$, the better. Our second metric, next to accuracy, is warmup length which is defined as the number of instructions required by the given warmup technique. Likewise, the smaller the warmup length, the smaller the total simulation time, the better.

6 Evaluation

We now evaluate the accuracy and warmup length of BHM compared to fixed-length warmup; section 6.1 covers accuracy and section 6.2 covers warmup length. Throughout this evaluation we consider a sampling unit size of 10K instructions. The reason is that, as mentioned in section 3, small sampling unit sizes suffer most from the lack of warmup; small sampling unit sizes will stress our warmup approach the most. All the results presented in this paper are for 50 sampling units.

Further, we assume that the number of global and local history bits equals 16 for the BHM approach in sections 6.1 and 6.2. Section 6.3 then studies the impact of the BHM history length on accuracy and warmup length.

6.1 Accuracy

Comparison against fixed-length warmup. Figure 4 evaluates the accuracy of BHM compared to fixed-length warmup. Both warmup techniques are budgeted a 1M warmup length per sampling unit, *i.e.*, both warmup techniques use the same warmup length. The four graphs in Figure 4 represent four different branch predictors, namely the gshare, local, bimodal and hybrid branch predictors. The $\Delta MPKIs$ are shown for both warmup techniques. We observe that BHM substantially outperforms fixed-length warmup. Over all four branch predictors, the average $\Delta MPKI$ decreases from 0.48 (under fixed-length warmup) to 0.29 (under BHM) which is 39% more accurate.

Comparison against MRRL. Figure 5 compares BHM against MRRL. As mentioned before, MRRL looks how far one needs to go back in the pre-sampling unit for encountering branch instances of the same static branch for all branch instances in the sampling unit. The results in Figure 5 show that BHM clearly outperforms MRRL. Over all four branch predictors, the average $\Delta MPKI$ decreases from 2.13 (under MRRL) to 0.29 (under BHM) which is 87% more accurate. The important difference between MRRL and BHM is that BHM, in contrast to MRRL, takes into account branch histories; this results in significantly more accurate branch predictor state warmup for BHM compared to MRRL. Note that MRRL also performs worse than fixed 1M warmup, compare Figure 4 against Figure 5. The reason is that, because of the fact that MRRL does not take

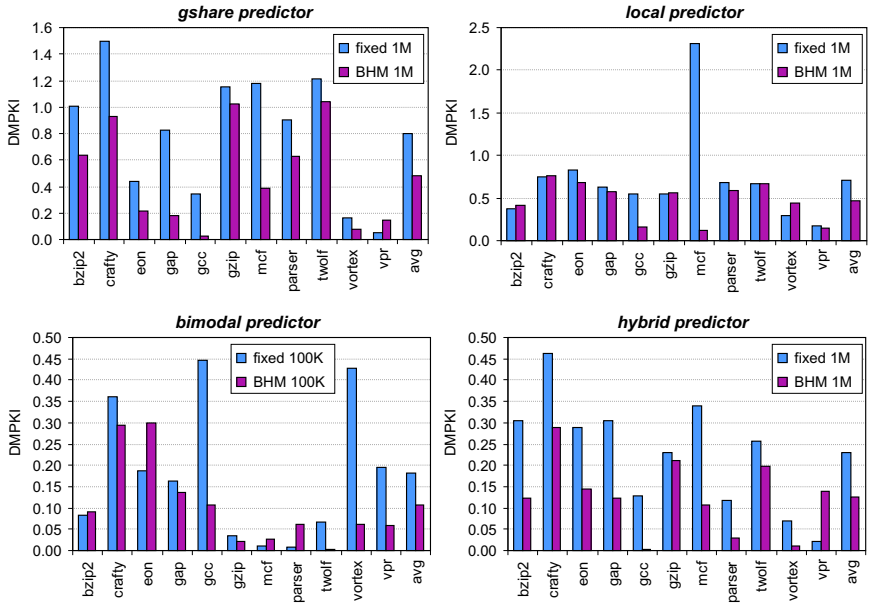

Fig. 4. $\Delta MPKI$ results for fi xed 1M warmup and BHM for the gshare, local, bimodal and hybrid branch predictors.

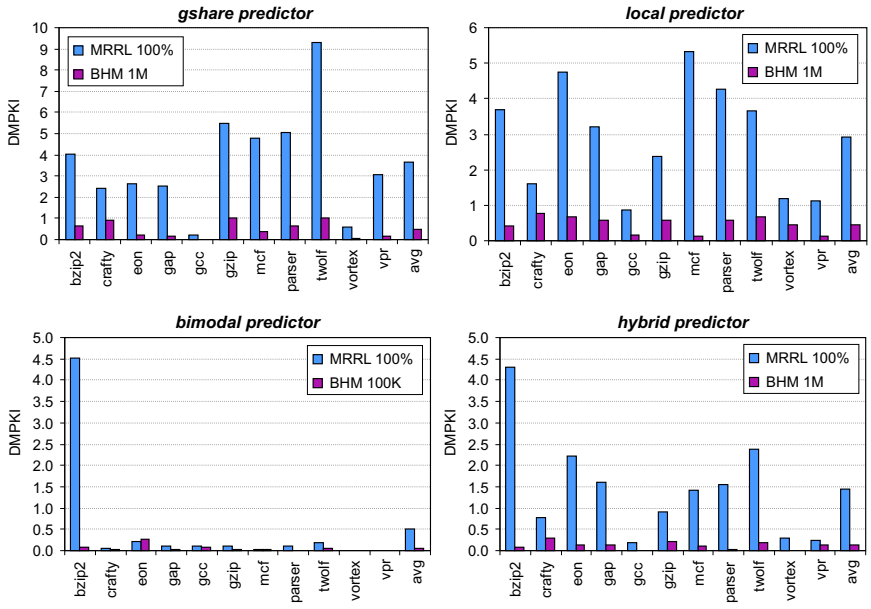

Fig. 5. $\Delta MPKI$ results for MRRL and BHM for the gshare, local, bimodal and hybrid branch predictors. For MRRL, we consider all branch instances in the sampling unit, hence the 'MRRL 100%' labels.)

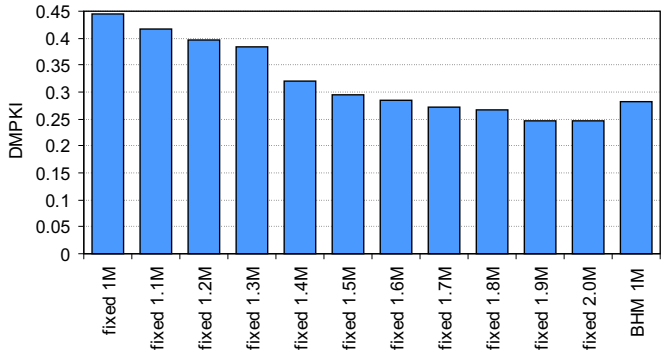

Fig. 6. Average $\Delta MPKI$ over the four branch predictors as a function of warmup length for the fixed-length warmup approach compared to BHM 1M.

into account branch history, MRRL is unable to come up with long enough warmup lengths for accurately warming up the branch predictors. The average warmup length through MRRL is only 200K instructions per sampling unit; according to our results, much larger warmup lengths are required to accurately warmup branch predictors.

6.2 Warmup length

In order to quantify the reduction in warmup length through BHM compared to fixed-length warmup, we have measured the average $\Delta MPKI$ over the four branch predictors as a function of warmup length, see Figure 6. The average $\Delta MPKI$ is shown for fixed-length warmup with the warmup budget varying between 1M and 2M instructions per sampling unit. The $\Delta MPKI$ for BHM with a 1M warmup length budget per sampling unit is shown on the right. We observe that fixed-length warmup achieves about the same accuracy as BHM for a warmup length of 1.6M instructions per sampling unit. In other words, BHM with a 1M warmup budget per sampling unit results in a 1.6X reduction in warmup length compared to fixed-length warmup while achieving the same accuracy.

Figure 7 shows MPKI versus warmup length for the `gcc` benchmark and the four branch predictors. Note that the horizontal axes are shown on a log scale. The two curves in each graph represent fixed-length warmup and BHM warmup, respectively; and the various points in these curves represent different warmup budgets. This graph clearly shows that BHM achieves the same accuracy with substantially shorter warmup lengths, or reverse, BHM achieves better accuracy for the same warmup length.

6.3 Impact of BHM history length

Note that the amount of branch history used by three of the four branch predictors, namely the gshare, local and hybrid predictors, equals 16 bits. The number of BHM history bits used for computing the warmup length also equals 16 bits. The question however is how sensitive BHM's accuracy is to the BHM history length.

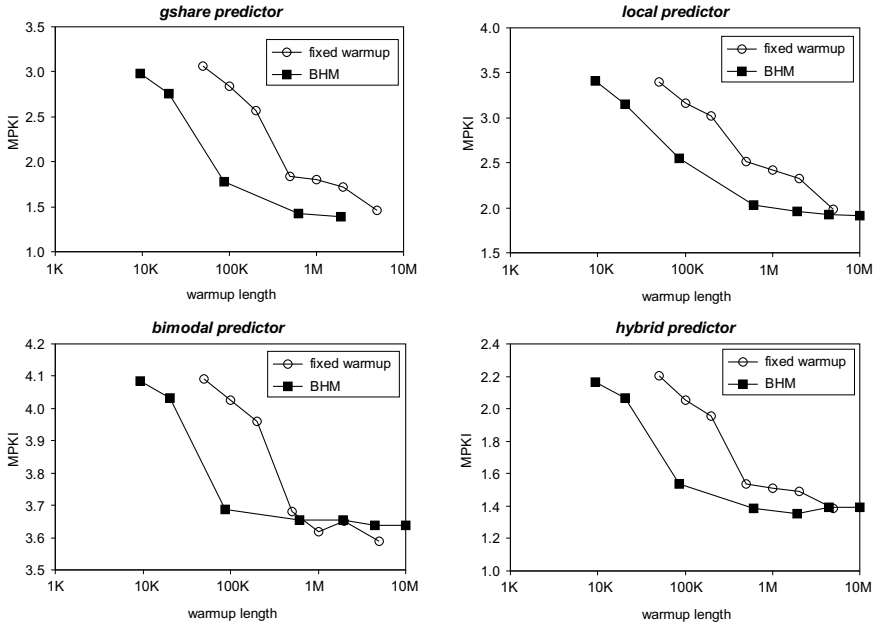

Fig. 7. Comparing BHM versus fixed warmup in terms of MPKI versus warmup length for gcc.

Figure 8 explores the impact of the BHM history length. The average $\Delta MPKI$ over all benchmarks is shown on the vertical axis versus the warmup length on the horizontal axis. The five curves represent the five branch predictors. The different points on each curve represent different BHM history lengths. We varied the BHM history length from 0, 2, 4, 8 to 16; when varying the history length we simultaneously vary the global and local BHM history lengths. A zero BHM history length means that no global and local history is taken into account for building the BHM distribution. In other words, the BHM warmup method then simply looks for the last occurrence of the same static branch for updating the BHM distribution. In all of these experiments, we budgeted a warmup length to 1M instructions per sampling unit.

There are two interesting observations to be made from this graph. First, accuracy improves or $\Delta MPKI$ decreases with increasing BHM history lengths. This is to be expected because the more history is taken into account, the better BHM will be able to determine how far it needs to go back in the pre-sampling unit for appropriate warmup. Second, small BHM histories are unable to budget the warmup lengths so that the average warmup length per sampling unit effectively equals the 1M instruction warmup budget. For example, a zero BHM history only yields slightly more than 200K instructions of warmup per sampling unit. In other words, it is impossible for BHM with limited history to fully exploit the available warmup budget. By increasing the BHM history length, BHM is better able to approach the target 1M warmup length per sampling unit. (Note that the MRRL approach [17,20] corresponds to a zero BHM history length.) We further observe that an 8 bit and a 16 bit BHM history length yields ap-

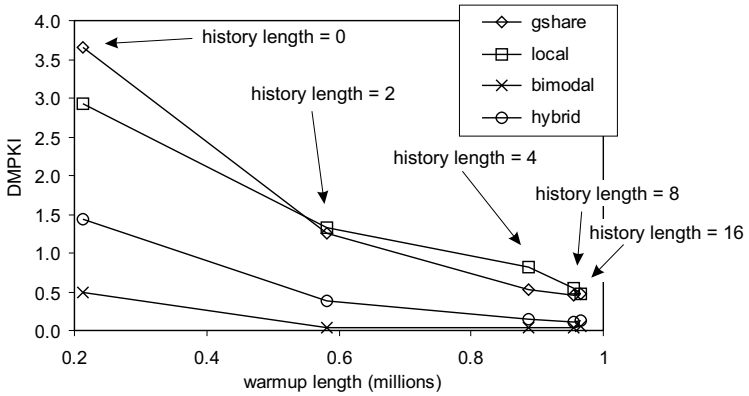

Fig. 8. Evaluating the impact of the BHM history length on accuracy and warmup length.

proximately the same accuracy. From this experiment, we thus conclude that in order to achieve accurate warmup for branch predictors, the BHM history length needs to be set to an appropriate value, for example, to the maximum history length one would look at during the branch predictor design space exploration.

7 Conclusion

Sampled simulation is a well known approach to speed up architectural simulations that are heavily used by computer architects and designers. An important issue with sampled simulation however is the cold-start problem, *i.e.*, the microarchitecture state is unknown at the beginning of each sampling unit. Although a great deal of work has been done on cache structure warmup, very little research has been done on branch predictor warmup.

This paper proposed Branch History Matching (BHM) as a novel branch predictor warmup method. The idea is to analyze the sampling unit as well as the pre-sampling unit for recurring branch instances of the same static branch with similar global and local branch histories. By doing so, BHM builds a distribution for each sampling unit that characterizes the branch locality behavior. BHM then budgets its total warmup budget to the various sampling units. Sampling units that are warmup-sensitive are budgeted more warmup; sampling units that are warmup-insensitive are budgeted less warmup. Compared to fixed-length warmup, BHM achieves better accuracy for the same total warmup budget, or reverse, BHM achieves the same accuracy with a shorter total warmup budget.

Acknowledgements

The authors would like to thank the anonymous reviewers for their valuable feedback. Lieven Eeckhout is a Postdoctoral Fellow with the Fund for Scientific Research—Flanders (Belgium) (FWO—Vlaanderen). This research is also supported by Ghent University, IWT, HiPEAC and the European SARC project No. 27648.

References

1. Conte, T.M., Hirsch, M.A., Menezes, K.N.: Reducing state loss for effective trace sampling of superscalar processors. In: ICCD. (1996) 468–477
2. Sherwood, T., Perelman, E., Hamerly, G., Calder, B.: Automatically characterizing large scale program behavior. In: ASPLOS. (2002) 45–57
3. Wunderlich, R.E., Wenisch, T.F., Falsafi , B., Hoe, J.C.: SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In: ISCA. (2003) 84–95
4. Yi, J.J., Kodakara, S.V., Sendag, R., Lilja, D.J., Hawkins, D.M.: Characterizing and comparing prevailing simulation techniques. In: HPCA. (2005) 266–277
5. Van Biesbrouck, M., Eeckhout, L., Calder, B.: Efficient sampling startup for sampled processor simulation. In: HiPEAC. (2005) 47–67
6. Wenish, T., Wunderlich, R., Falsafi , B., Hoe, J.: TurboSMARTS: Accurate microarchitecture simulation in minutes. In: SIGMETRICS. (2005) 408–409
7. Barr, K.C., Pan, H., Zhang, M., Asanovic, K.: Accelerating multiprocessor simulation with a memory timestamp record. In: ISPASS. (2005) 66–77
8. Van Ertvelde, L., Hellebaut, F., Eeckhout, L., De Bosschere, K.: NSL-BLRL: Efficient cache warmup for sampled processor simulation. In: ANSS. (2006) 168–175
9. Wenisch, T.F., Wunderlich, R.E., Falsafi , B., Hoe, J.C.: Simulation sampling with live-points. In: ISPASS. (2006) 2–12
10. Barr, K.C., Asanovic, K.: Branch trace compression for snapshot-based simulation. In: ISPASS. (2006) 25–36
11. Girbal, S., Mouchard, G., Cohen, A., Temam, O.: DiST: A simple, reliable and scalable method to significantly reduce processor architecture simulation time. In: SIGMETRICS. (2003) 1–12
12. Lauterbach, G.: Accelerating architectural simulation by parallel execution of trace samples. Technical Report SMLI TR-93-22, Sun Microsystems Laboratories Inc. (1993)
13. Kessler, R.E., Hill, M.D., Wood, D.A.: A comparison of trace-sampling techniques for multi-megabyte caches. IEEE Transactions on Computers **43** (1994) 664–675
14. Wood, D.A., Hill, M.D., Kessler, R.E.: A model for estimating trace-sample miss ratios. In: SIGMETRICS. (1991) 79–89
15. Conte, T.M., Hirsch, M.A., Hwu, W.W.: Combining trace sampling with single pass methods for efficient cache simulation. IEEE Transactions on Computers **47** (1998) 714–720
16. Haskins Jr., J.W., Skadron, K.: Minimal subset evaluation: Rapid warm-up for simulated hardware state. In: ICCD-2001. (2001) 32–39
17. Haskins Jr., J.W., Skadron, K.: Memory Reference Reuse Latency: Accelerated warmup for sampled microarchitecture simulation. In: ISPASS. (2003) 195–203
18. Eeckhout, L., Luo, Y., De Bosschere, K., John, L.K.: BLRL: Accurate and efficient warmup for sampled processor simulation. The Computer Journal **48** (2005) 451–459
19. Luo, Y., John, L.K., Eeckhout, L.: Self-monitored adaptive cache warm-up for microprocessor simulation. In: SBAC-PAD. (2004) 10–17
20. Haskins, J.W., Skadron, K.: Accelerated warmup for sampled microarchitecture simulation. ACM Transactions on Architecture and Code Optimization (TACO) **2** (2005) 78–108
21. Perelman, E., Hamerly, G., Calder, B.: Picking statistically valid and early simulation points. In: PACT. (2003) 244–256
22. Srivastava, A., Eustace, A.: ATOM: A system for building customized program analysis tools. Technical Report 94/2, Western Research Lab, Compaq (1994)
23. McFarling, S.: Combining branch predictors. Technical Report WRL TN-36, Digital Western Research Laboratory (1993)
24. Yeh, T.Y., Patt, Y.N.: Alternative implementations of two-level adaptive branch prediction. In: ISCA. (1992) 124–134