NSL-BLRL: Efficient Cache Warmup for Sampled Processor Simulation

Luk Van Ertvelde Filip Hellebaut Lieven Eeckhout Koen De Bosschere

ELIS Department, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Email: {luk.vanertvelde,filip.hellebaut,leeckhou,kdb}@elis.UGent.be

Abstract

Architectural simulation is extremely time-consuming given the huge number of instructions that need to be simulated for contemporary benchmarks. Sampled simulation which selects a number of samples from the complete benchmark execution yields substantial speedups. However, there is one major issue that needs to be dealt with in order to minimize non-sampling bias, namely the hardware state at the beginning of each sample. This is well known in the literature as the cold-start problem. The hardware structures that suffer the most from the cold-start problem are cache hierarchies.

In this paper we propose NSL-BLRL which combines two previously proposed cache hierarchy warmup approaches, namely No-State-Loss (NSL) and Boundary Line Reuse Latency (BLRL). The idea of NSL-BLRL is to warmup the cache hierarchy using a hardware state checkpoint that stores a truncated NSL stream. The NSL stream is a least-recently used stream of (unique) memory references in the pre-sample. This NSL stream is then truncated to form the NSL-BLRL warmup checkpoint; this is done by inspecting the sample for determining how far in the pre-sample one needs to go back to accurately warmup the hardware state for the given sample. We show using SPEC CPU2000 benchmarks that NSL-BLRL is (i) nearly as accurate as BLRL and NSL for sampled processor simulation, (ii) yields simulation time speedups of several orders of magnitude compared to BLRL, and (iii) is more space-efficient than NSL. As such, we conclude that NSL-BLRL is a highly efficient and accurate cache warmup strategy for sampled processor simulation.

1 Introduction

Current microarchitectural research and microprocessor development relies heavily on cycle-level architectural simulations. Cycle-level simulations model a microarchitecture at a fairly detailed level while executing real-life applications. The price paid for such detailed simulations of real-life benchmarks obviously is simulation speed. Simulating a full benchmark execution can take days or even weeks for completion. If we take into account that during microarchitectural research and microprocessor development 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 during design space exploration is infeasible.

Several approaches have been proposed in the recent literature to address this issue. One particular proposal is *sampled simulation* [4, 5, 12, 15, 16, 19]. Sampled simulation selects a number of execution intervals from a complete benchmark execution, called samples, to be simulated. 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. The pre-sample is the sequence of contiguous instructions before the sample, *i.e.*, from the end of the previous sample until the beginning of the current sample. This is to *warmup* large hardware structures so that the hardware state at the beginning of the sample is a close estimate of what a detailed simulation would reach at the beginning of the sample in case the full benchmark would have been simulated. Due to the extremely long history in microarchitectural state (for example 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. Several warmup proposals have been made in the literature. No-State-Loss (NSL) [4], Memory Hierarchy State (MHS) [17], Memory Reference Reuse Latency (MRRL) [10] and Boundary Line Reuse Latency (BLRL) [8] are the most accurate and flex-

ible approaches existing today. In this paper we propose a new warmup approach that combines NSL with BLRL, called NSL-BLRL, which substantially outperforms existing warmup strategies in terms of warmup length and disk space requirements. The efficiency of NSL-BLRL is evaluated using SPEC CPU2000 benchmarks.

This paper makes the following contributions:

- We show that No-State-Loss (NSL) can be combined with Boundary Line Reuse Latency (BLRL) into an efficient warmup strategy called NSL-BLRL for cache hierarchies in sampled processor simulation. NSL-BLRL outperforms previously proposed warmup strategies. The NSL-BLRL approach that we propose is more than two orders of magnitude more efficient in terms of warmup memory references compared to the best performing contiguous warmup approaches such as BLRL while achieving the same accuracy. Compared to NSL, NSL-BLRL requires 30% less disk storage. This is an important issue when numerous samples along with their warmup info need to be stored on disk; the total amount of disk space requirements might become very large. In addition, limiting the size of the warmup info stored on disk also reduces simulation time; reading the warmup info from disk and transfering over a network for the parallel simulation of the various samples on a cluster of machines, can be done substantially faster.

- This paper extends our previous work [6] in two ways. Our previous work showed that NSL can be combined with MRRL; here we show that NSL can be combined with BLRL into a warmup strategy NSL-BLRL that outperforms NSL-MRRL in both accuracy and efficiency. Secondly, we show in this paper that NSL-BLRL is also applicable to sampled processor simulation; our previous work on combining NSL with MRRL focused on cache simulation only, processor simulation was not considered there.

2 Sampled processor simulation

In sampled processor simulation, a number of samples are chosen from a complete benchmark execution. The instructions between two samples are called the *pre-sample*. Sampled simulation only uses 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. Several approaches have been described in the recent literature to select such samples: random sampling by Conte *et al.* [4], profile-driven sampling by scaling the basic block execution counts by Dubey and Nair [5], periodic selection as done in SMARTS [19], selection based on clustering similarly behaving intervals as done by Lafage and Seznec [12] as well as in SimPoint [15, 16].

The second issue next to the selection of representative samples 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 from the pre-sample are not simulated during sampled processor simulation. Several techniques have been proposed in the literature to address this important issue. Most of these use a number of instructions preceding the sample to *warmup* hardware state before each sample [4, 9, 11, 14, 18]. Under such a warmup strategy, sampled simulation consists of three steps. The first step is *cold simulation* in which the program execution is fast-forwarded, *i.e.*, functional simulation without updating microarchitectural state. The second step is *warm simulation* which updates microarchitectural state. This is typically done for large hardware structures such as caches, TLBs, branch predictors, etc. Under warm simulation, no performance metrics are calculated. The warm simulation phase can be very long since 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, etc. These three steps are repeated for each sample.

3 Warmup strategies

We now discuss a number of recently proposed warmup strategies that are fairly accurate and efficient, namely MRRL, BLRL and a number of hardware state checkpointing techniques including NSL.

3.1 Memory Reference Reuse Latency (MRRL)

Haskins and Skadron [10] propose Memory Reference Reuse Latency (MRRL) for accurately warming up 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 a reference to address A and the next reference to A. MRRL basically computes the distribution of the reuse latencies over the pre-sample as well as the sample. Once the distribution of reuse latencies is computed, they then compute the reuse latency that corresponds to $K\%$ of the cumulative distribution, say w. In MRRL, warmup then starts w instructions prior to 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, that 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. Reverse, 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 the percentile $K\%$ large enough. The result is that the warmup will be longer than needed for the attained accuracy.

3.2 Boundary Line Reuse Latency (BLRL)

Boundary Line Reuse Latency (BLRL) [7, 8] is quite 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. A distribution is then measured for the pre-sample reuse latency, similar to what is done in MRRL. Also very much like what happens in MRRL, the warmup length w is then determined using a percentile $K\%$.

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 originating from instructions in both the pre-sample *and* sample. Second, BLRL only considers 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. Previous work [8] has shown that BLRL substantially outperforms MRRL; the warmup length of BLRL is nearly half the warmup length of MRRL for the same level of accuracy.

3.3 Hardware state checkpointing

Another approach to the cold-start problem is to *checkpoint* or to store the hardware state at the beginning of each sample and impose this state during sampled simulation. This approach yields perfectly warmed up hardware state. However, the storage needed to store these checkpoints can explode in case many samples are required. 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.

Since this is infeasible to do in practice, researchers have proposed more efficient approaches to hardware state checkpointing. One example is the No-State-Loss (NSL) approach [3, 13] which scans the pre-sample and records the latest reference to each unique memory location in the pre-sample. The obtained stream is called the *least recently used (LRU) stream*. For example, the LRU stream of the following reference stream 'ABAACDABA' is 'CDBA'. The LRU stream can be computed by building the LRU stack for the given reference stream. It is easily understandable that both reference streams, the original reference stream as well as the LRU stream, yield the same state when applied to an LRU stack. The no-state-loss warmup method exploits this property by computing the LRU stream of the pre-sampling unit and applying this stream to the cache as warmup. By consequence, the no-state-loss warmup strategy yields perfect warmup for caches with an LRU replacement policy. Barr *et al.* [1] extended this approach for reconstructing the cache and directory state during sampled multiprocessor simulation.

Van Biesbrouck *et al.* [17] proposed the Memory Hierarchy State (MHS) approach. In MHS, the largest cache of interest is simulated once for each sample. The cache's content is then stored on disk. The content of smaller-sized caches can then be derived from the checkpoint. The disadvantage of this approach compared to NSL is that MHS requires the cache line size to be fixed. Whenever a cache needs to be simulated with a different cache line size, the warmup info needs to be recomputed. NSL does not have this disadvantage. The advantage of MHS over NSL however is that it is more space efficient, *i.e.*, a smaller disk space is required for storing the warmup info. The reason is that NSL stores all unique pre-sample memory references; MHS on the other hand, discards conflicting memory references from the warmup info for a given maximum cache size. A second advantage of MHS over NSL is that computing the MHS warmup is done faster than computing the NSL warmup info; NSL does an LRU stack simulation whereas MHS only simulates one particular cache configuration.

Our NSL-BLRL approach could be viewed as a form of hardware state checkpointing. When comparing NSL-BLRL against existing hardware state checkpointing techniques we conclude that (i) NSL-BLRL is more space-efficient than NSL, *i.e.*, requires less disk space than NSL, and (ii) NSL-BLRL is more broadly applicable during design space exploration than MHS because the NSL-BLRL warmup info is independent of the cache block size.

Note that efficient architectural checkpointing tech-

niques can be implemented for cache hierarchies including TLB structures. However, architectural checkpointing techniques for branch predictors that can be re-used over a number of branch predictors is not easy to do. Therefore, researchers have proposed a pragmatic approach by storing the contents of the branch predictors of interest as a checkpoint.

4 NSL-BLRL: combining NSL and BLRL

This paper proposes to combine the no-state-loss (NSL) warmup method with BLRL into NSL-BLRL. This is done by computing both the LRU stream as well as the BLRL warmup buckets corresponding to a given percentile $K\%$. Only the unique references that are within the warmup buckets will be used to warmup the caches. This could be viewed as pruning the LRU stream with BLRL information. Reverse, this method could also be viewed as selecting the LRU stream from the BLRL warmup buckets. Using NSL-BLRL as a warmup approach then works as follows. The reduced LRU stream as it is obtained through NSL-BLRL is to be stored on disk as a hardware state checkpoint. Upon simulation of a sampling unit, the reduced LRU stream is then loaded from disk, the cache state is warmed up and finally, the simulation of the sample gets started. The advantage over NSL is that NSL-BLRL requires less disk space to store the warmup memory references; in addition, the smaller size of the reduced LRU stream results in faster warmup processing. The advantage over BLRL is that loading the reduced LRU stream from disk is more efficient than the warm simulation needed for BLRL. According to our results, the warmup length for BLRL is at least two orders of magnitude longer than for NSL-BLRL. As such, significant speedups are to be obtained compared to BLRL. Note that NSL-BLRL inherits the limitation from NSL of only guaranteeing perfect warmup for caches with LRU replacement. Caches with other replacement policies such as random, first-in first-out (FIFO), not-most-recently-used (NMRU) are not guaranteed to get a perfectly warmed up cache state under NSL-BLRL (as is the case for NSL)—however, we believe the difference in warmed up hardware state would be very small.

5 Experimental setup

For the evaluation we use 9 SPEC CPU2000 integer benchmarks[1]; see Table 1. The binaries which were compiled and optimized for the Alpha 21264 processor, were taken from the SimpleScalar website[2]. All measurements presented in this paper are obtained using the MRRL software[3] which on its turn is based on the SimpleScalar software [2]. The baseline processor simulation model is given in Table 2. The caches use write-allocate and write-back policies. We consider 50 samples (each containing 1M instructions). We select a sample every 100M instructions unless mentioned otherwise. 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 using the IPC prediction error, *i.e.*, the procentual difference between the IPC for perfect warmup against the IPC for the warmup strategy of interest. A positive error means a IPC overestimation of the warmup approach compared to the

benchmark	input
bzip2	program
crafty	ref
eon	rushmeier
gcc	integrate
gzip	graphic
parser	ref
twolf	ref
vortex	lendian2
vpr	route

Table 1. The SPEC CPU2000 integer benchmarks used in this study along with their input (all inputs are reference inputs).

L1 I-cache	16KB, 2-way set-assoc
L1 D-cache	32KB, 4-way set-assoc
L2 cache	1MB, 4-way set-assoc, 20 cycles access lat
I-TLB and D-TLB	32-entry 8-way set-assoc with 4KB pages
memory	150 cycle round trip access
branch predictor	8K-entry table hybrid predictor
branch miss penalty	14 cycles
IFQ	32-entry instruction fetch queue
RUU and LSQ	128 entries and 32 entries, respectively
processor width	8 wide
functional units	8 intALUs, 4 ld/st, 2 fp, 2 int and 2 fp mult/div

Table 2. Baseline processor simulation model.

[1] `http://www.spec.org`
[2] `http://www.simplescalar.com`
[3] `http://www.cs.virginia.edu/~jwh6q/mrrl-web/`

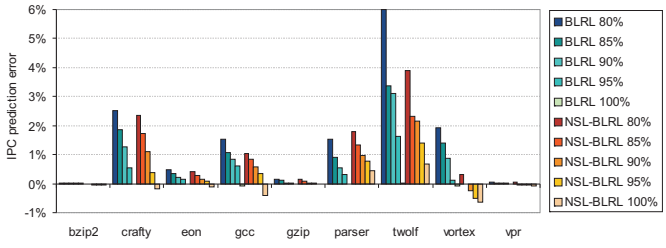

Figure 1. IPC prediction error for BLRL versus NSL-BLRL.

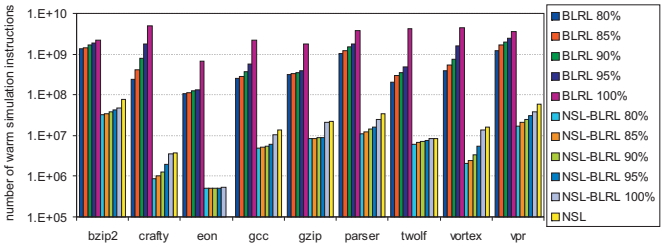

Figure 2. The number of warm simulation instructions for BLRL versus the number of warm simulation references for NSL-BLRL.

perfect warmup case.

6 Results

In this section, we extensively evaluate our NSL-BLRL approach and compare it against NSL and BLRL. We have a number of criteria to evaluate our improved warmup proposal, namely accuracy, number of warm simulation instructions, overall simulation and the amount of storage requirements.

6.1 Accuracy

Our first criterion to evaluate NSL-BLRL is its accuracy. Figure 1 shows the IPC prediction error for BLRL, NSL and NSL-BLRL for the various benchmarks and for varying percentiles K%. (Note that NSL yields the same accuracy as NSL-BLRL 100%.) The IPC prediction error is the relative error compared to a full warmup run, *i.e.*, all instructions prior to the sample are simulated. In the IPC prediction errors that we present here, we assume that there is no stale state (no stitch) when warming up the hardware state before simulating a sample. This is to stress the warmup techniques; in addition, this is also the error one would observe under checkpointed parallel sampled simulation. A number of comments and observations need to be made here. As reported in previous work, BLRL results in a highly accurate warmup. BLRL yields small IPC prediction errors of only a few percent. Especially for large percentiles $K\%$, the IPC prediction error due to incorrect hardware state is very small. For example, for BLRL 95%, the maximum error is only 1.6% (twolf). For BLRL 100%, the error is almost zero. Comparing NSL-BLRL versus BLRL for a given percentile $K\%$ typically gives slightly higher IPC prediction errors, however, the difference is very small (less than 1%). There are two reasons for these slightly higher IPC prediction errors. First, NSL-BLRL only warms the cache state but does not warm branch predictor state. BLRL on the other hand warms both the cache hierarchy and branch predictor state. However, we found this influence to be very small. To experimentally verify this, we compared the accuracy of NSL-BLRL versus BLRL for perfect branch predictors—this was to exclude the branch predictor component in the warmup state—and we obtained very similar results to what is being reported here in Figure 1. As such, we conclude that the impact of the branch predictor state is very small. The second reason for the difference between the NSL-BLRL and BLRL is that while warming the caches through NSL-BLRL we do not keep track of dirty cache blocks, whereas BLRL does keep track of dirty cache blocks. Our results show that not warming dirty cache block info only has a small impact on overall accuracy. This is to be expected given the fact that contemporary out-of-order microprocessors give priority to load operations over writing back dirty data to upper layers of the memory hierarchy. However, if warming dirty cache blocks needs to be supported, extending our framework for supporting this would not be difficult.

In summary, we can conclude that NSL-BLRL is a highly accurate cache warmup approach that is nearly as accurate as BLRL. Especially, high percentiles $K\%$ yield highly accurate performance estimates. The maximum error for $K = 95\%$ equals 1.4% (twolf); for $K = 100\%$, the maximum error is even less, 0.66%.

6.2 Warmup length

We now compare the number of warm simulation instructions that need to be processed. Figure 2 shows the number of warm simulation instructions for BLRL as well as the number of warm simulation references for NSL-BLRL for different percentiles $K\%$. Note that the vertical axis is on a logarithmic scale. We observe that NSL-BLRL yields a reduction in the number of warm simulation instructions by two to three orders of magnitude compared to BLRL. The reason for this dramatic reduction is that the number of warm simulation instructions for NSL-BLRL is proportional to the number of unique references in the pre-sample. BLRL on the other hand, uses all references from a given warmup starting point up to the sample starting point.

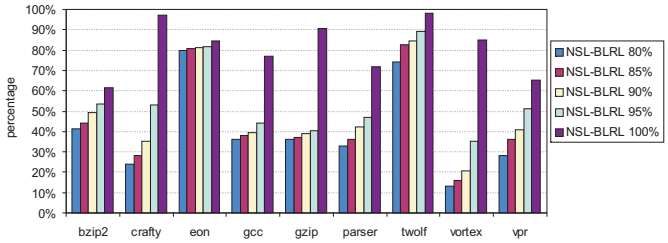

Figure 3. The number of warm simulation instructions for NSL-BLRL as a fraction of the number of warm simulation instructions for NSL.

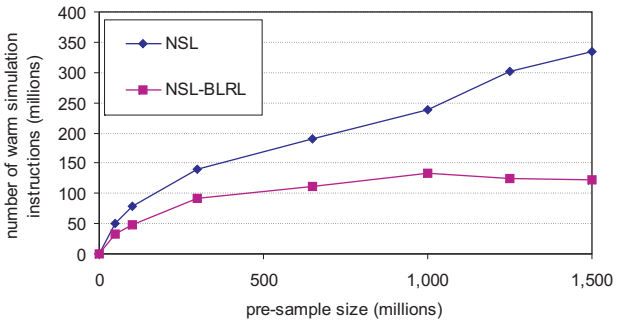

Figure 4. The number of warm simulation instructions for NSL-BLRL and NSL as a function of the pre-sample size. This is for bzip2**.**

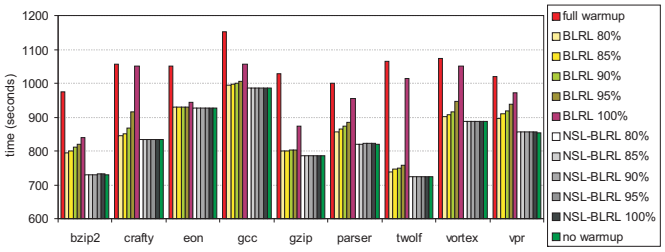

Figure 5. Simulation time for BLRL and NSL-BLRL in case of sampled simulation using fastforwarding.

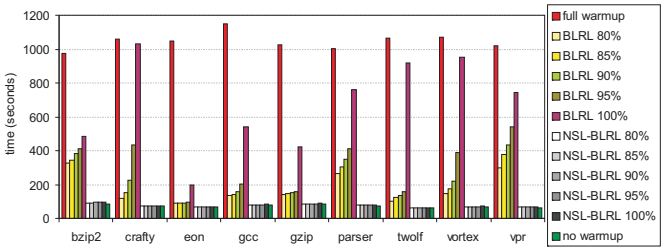

Figure 6. Simulation time for BLRL and NSL-BLRL in case of checkpointed sampled simulation.

Note that these results were obtained for 100M instruction pre-samples prior to each sample. For larger pre-samples, the difference in the number of warm simulation instructions will even increase when comparing BLRL versus between NSL-BLRL.

Comparing now NSL-BLRL versus NSL we also observe a substantial decrease in the number of warm simulation instructions. Figure 3 shows the number of warm simulation instructions of NSL-BLRL as a fraction of NSL. Some benchmarks do not benefit substantially from NSL-BLRL compared to NSL. However, we observe that NSL-BLRL 100% yields substantial warm simulation reductions for some benchmarks—up to 39% for bzip2; *i.e.*, the warmup length for NSL-BLRL 100% is 61% of the NSL warmup length. For smaller $K\%$ percentiles the reduction in warmup length increases significantly. Again, these numbers are given for a 100M instruction pre-sample. For larger pre-sample sizes, the benefit for NSL-BLRL over NSL in terms of the number of warm simulation instructions even increases. This is illustrated in Figure 4 where the number of warm simulation instructions is shown as a function of the pre-sample size for NSL and NSL-BLRL for bzip2. Similar curves were obtained for other benchmarks. The important trend to be observed from this graph is that the number of warm simulation instructions does not increase that fast for NSL-BLRL as it does for NSL. As such, we can conclude that NSL-BLRL is better scalable for larger pre-sample sizes and thus, longer running applications.

6.3 Simulation time

The number of warm simulation instructions only gives a vague idea of what the overall simulation time speedup would be for NSL-BLRL compared to BLRL. We identify two scenarios for sampled simulation, namely using fast-forwarding to navigate between samples, and checkpointing to restore machine state at the beginning of each sample. We first consider the case where fast-fowarding is used to go from one sample to the next sample. In this scenario, cold simulation is done through fast-forwarding. When the warm simulation starting point is reached for BLRL, warm simulation gets started until the beginning of the sample is reached. Then, simulation switches to hot simulation. For NSL-BLRL, cold simulation is done until the beginning of the sample, then the NSL-BLRL checkpoint is loaded from disk and the hardware state is updated. Once the hardware state is updated, hot simulation of the sample gets started. The results in Figure 5 show the simulation time in seconds under fast-forwarding. We observe that BLRL achieves a substantial simulation time reduction compared to full

warmup. NSL-BLRL reduces the overall simulation time even further, even onto a level where warmup using NSL-BLRL is nearly as fast as no-warmup. In other words, the cost for warming up hardware state under fast-forwarding is nearly zero under NSL-BLRL. Note also that different percentiles $K\%$ do not affect overall simulation time. As such, we could conclude that a percentile $K = 100\%$ is the optimal choice since it gives the highest accuracy while incurring no additional simulation time overhead compared to smaller percentiles $K\%$.

We now consider checkpointing instead of fast-forwarding for jumping between the various samples. Under checkpointed sampled simulation, there is no cold simulation. Simulating a sample starts by loading a machine state checkpoint from disk and initiating the warm simulation. Under BLRL, the warm simulation phase involves warming up caches while functionally simulating all instructions prior to the sample. Under NSL and NSL-BLRL, warm simulation involves loading a machine state checkpoint. Once the machine state is updated, hot simulation gets started. Under checkpointed sampled simulation we obtain the simulation time results presented in Figure 6. BLRL yields substantial simulation time reductions over full warmup. Note that the simulation time reductions under checkpointing are even bigger than under fast-forwarding. This is to be expected as checkpointed simulation does not require cold simulation opposed to fast-forwarding. Another interesting note is that the simulation time reduction when comparing NSL-BLRL versus BLRL under checkpointing is higher than under fast-forwarding. Under fast-forwarding, NSL-BLRL achieves a reduction in simulation time over BLRL up to a factor 1.4X; under checkpointing, NSL-BLRL achieves a 2.9X up to 14.9X simulation time speedup over BLRL. This is to be explained for the same reason; checkpointed simulation does not involve cold simulation.

6.4 Storage

We now quantify the storage requirements of NSL-BLRL for storing the hardware state checkpoints on disk. Figure 7 shows the amount of storage requirements for NSL-BLRL compared to NSL. (Note that BLRL does not require any significant storage.) The numbers shown in Figure 7 represent the number of KBs of storage needed to store one hardware state checkpoint in compressed format. For NSL, the average storage requirement per sample is 810KB; the maximum observed is for bzip2, 2.5MB. For NSL-BLRL, the storage requirements are greatly reduced compared to NSL. For example, for $K = 100\%$, the average storage requirement is 553KB (a 32% reduction); for $K = 95\%$, the average storage requirement is 425KB (a 48% reduction). As such, we conclude that the real benefit of NSL-BLRL compared to NSL is its reduced storage requirements. (Recall that NSL-BLRL and NSL are comparable in terms of accuracy and simulation time.) In case a large number of checkpoints need to be stored on disk for a complete benchmark suite, then we can easily end up with thousands of samples and corresponding checkpoint files. For example, for SimPoint there are 7,392 1M instruction samples for the whole SPEC CPU2000 benchmark suite[4]. If 810KB needs to be stored on disk per sample, then approximately 6GB disk space is required for storing the NSL hardware state warmup info. Note that this is an optimistic approximation. In our experimental setup we assumed 100M instruction pre-samples. Larger pre-samples will results in even larger NSL warmup checkpoints to be stored on disk, as discussed previously, see also Figure 4. As such, the total storage requirements are expected to be substantially larger than the 6GB mentioned above. In addition, machine state checkpoints need to be stored on disk as well. Even though disks are cheap these days, maintaining such large checkpoint files might be impractical to do. We conclude that NSL-BLRL is capable of reducing the total disk space requirements for hardware state checkpointing by at least 30% without any loss in accuracy.

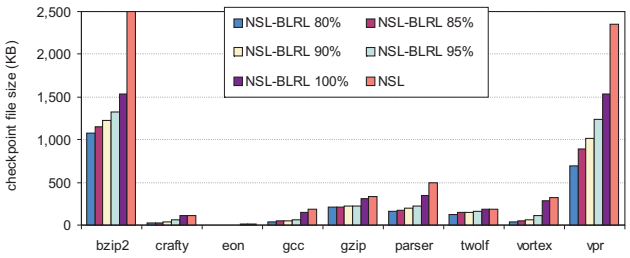

Figure 7. Storage requirements for NSL-BLRL compared to NSL: average number of KBs of disk storage needed for storing one hardware state checkpoint in compressed format.

7 Conclusions

Sampled simulation is an important tool for computer architecture research and development. The idea behind sampled simulation is to select a well chosen number of samples from a complete program execution. There are two major issues related to sampled simulation, (i) the selection of representative samples and (ii) warming up the correct hardware state at the beginning of each sample, well known as the cold-start problem.

This paper proposed to combine No-State-Loss (NSL) with Boundary Line Reuse Latency (BLRL) in a new

[4] http://www.cs.ucsd.edu/ calder/simpoint/

warmup strategy called NSL-BLRL. The basic idea is to truncate the NSL stream of memory references in a pre-sample using BLRL information. The NSL stream is the least recently used sequence of memory references in the pre-sample. BLRL then selects a fraction of this NSL stream based on how far back warmup needs to go in the pre-sample to accurately warmup the hardware state for the given sample. The NSL-BLRL warmup info could then be viewed as a hardware state checkpoint. Warming up a cache hierarchy using NSL-BLRL is then done by loading the checkpoint from disk and warming the caches using the NSL-BLRL reference stream. Compared to other existing hardware state checkpointing techniques, NSL-BLRL is more flexible in the sense that the warmup info can be used for a broader range of hardware configurations. For example, whereas Memory Hierarchy State (MHS) requires a fixed cache block size, NSL-BLRL does not.

Our experimental results using SPEC CPU2000 benchmarks show that NSL-BLRL is substantially faster than BLRL. In other words, the number of warmup instructions is reduced up to three orders of magnitude. NSL-BLRL is nearly as accurate as BLRL. The small deviation is due to not modeling dirty cache lines in NSL-BLRL but we found this difference to be very small. The shorter warmup length for NSL-BLRL results in substantial simulation speedups against BLRL. Under fast-forwarding, the simulation speedup is up to 1.4X. Under checkpointing, the simulation speedup varies between 2.9X and 14.9X. Compared to NSL, the real benefit of NSL-BLRL is in the reduced checkpoint files that need to be stored on disk. (In terms of accuracy and simulation time, NSL-BLRL is nearly as efficient as NSL.) NSL-BLRL typically yields 30% smaller hardware state checkpoint files which is extremely important when it comes to storing a large number of checkpoint files on disk for a large number of samples.

Acknowledgements

Filip Hellebaut is supported by a grant from the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT). Lieven Eeckhout is a Postdoctoral Fellow with the Fund for Scientific Research—Flanders (Belgium) (FWO—Vlaanderen). This research is also supported by Ghent University and the HiPEAC Network of Excellence.

References

[1] K. C. Barr, H. Pan, M. Zhang, and K. Asanovic. Accelerating multiprocessor simulation with a memory timestamp record. In *ISPASS*, pages 66–77, Mar. 2005.

[2] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set. Computer Architecture News, 1997. See also `http://www.simplescalar.com` for more information.

[3] T. M. Conte, M. A. Hirsch, and W. W. Hwu. Combining trace sampling with single pass methods for efficient cache simulation. *IEEE Transactions on Computers*, 47(6):714–720, June 1998.

[4] T. M. Conte, M. A. Hirsch, and K. N. Menezes. Reducing state loss for effective trace sampling of superscalar processors. In *ICCD*, pages 468–477, Oct. 1996.

[5] P. K. Dubey and R. Nair. Profile-driven sampled trace generation. Technical Report RC 20041, IBM Research Division, T. J. Watson Research Center, Apr. 1995.

[6] L. Eeckhout and K. De Bosschere. Yet shorter warmup by combining no-state-loss and MRRL for sampled LRU cache simulation. *Journal of Systems and Software*, 2006. Accepted for publication.

[7] L. Eeckhout, S. Eyerman, B. Callens, and K. De Bosschere. Accurately warmed-up trace samples for the evaluation of cache memories. In *Proceedings of the 2003 High Performance Computing Symposium (HPC-2003)*, pages 267–274, Apr. 2003.

[8] L. Eeckhout, Y. Luo, K. De Bosschere, and L. K. John. BLRL: Accurate and efficient warmup for sampled processor simulation. *The Computer Journal*, 48(4):451–459, May 2005.

[9] J. W. Haskins Jr. and K. Skadron. Minimal subset evaluation: Rapid warm-up for simulated hardware state. In *ICCD*, pages 32–39, Sept. 2001.

[10] J. W. Haskins Jr. and K. Skadron. Memory Reference Reuse Latency: Accelerated warmup for sampled microarchitecture simulation. In *ISPASS*, pages 195–203, Mar. 2003.

[11] R. E. Kessler, M. D. Hill, and D. A. Wood. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Transactions on Computers*, 43(6):664–675, June 1994.

[12] T. Lafage and A. Seznec. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In *WWC held in conjunction with ICCD*, Sept. 2000.

[13] G. Lauterbach. Accelerating architectural simulation by parallel execution of trace samples. Technical Report SMLI TR-93-22, Sun Microsystems Laboratories Inc., Dec. 1993.

[14] A.-T. Nguyen, P. Bose, K. Ekanadham, A. Nanda, and M. Michael. Accuracy and speed-up of parallel trace-driven architectural simulation. In *IPPS*, pages 39–44, Apr. 1997.

[15] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *PACT*, pages 244–256, Sept. 2003.

[16] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *ASPLOS*, pages 45–57, Oct. 2002.

[17] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Efficient sampling startup for sampled processor simulation. In *HiPEAC*, pages 47–67, Nov. 2005.

[18] D. A. Wood, M. D. Hill, and R. E. Kessler. A model for estimating trace-sample miss ratios. In *SIGMETRICS*, pages 79–89, May 1991.

[19] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In *ISCA-30*, pages 84–95, June 2003.