Approaching a High-Performance, General-Purpose Multi-Threaded Sampling Methodology

Alen Sabu[1], Harish Patil[2], Wim Heirman[2], Alexander Isaev[2], and Trevor E. Carlson[1]

[1]*National University of Singapore*
[2]*Intel Corporation*

1. Introduction

Computer architects rely heavily on microarchitectural simulation for processor state exploration and evaluation. Simulations allow the estimation of future system performance and efficiency using the present hardware. However, traditional cycle-accurate microarchitecture simulation is extremely slow, especially when evaluating workloads with many threads. An OpenMP application from the SPEC CPU 2017 benchmark suite with reference inputs takes weeks or months to simulate even on fast simulators like ZSim [**?**], Sniper [**?**], etc. As the complexity of the target system increases, the simulations take even longer when compared to native execution. One solution to this issue is to sample the whole application to estimate the performance. Since the application behavior exhibits repetitive behavior, sampling can effectively reduce the size of the application to a small fraction of its original, and the performance of the entire application can be extrapolated because of this repetitive behavior. The performance of the whole application can be extrapolated, considering the performance of only a few samples. SimPoint [**?**] uses the information about dynamic basic-block distribution to find phases in an application and select representative samples. SMARTS [**?**] is another such methodology that has alternate fast-forward intervals between the detailed simulation of sampling units.

Single-threaded sampling is now considered a solved problem. There are some solutions proposed for multithreaded sampling. Time-based sampling [**?**] [**?**] is a generic technique that considers time as the sampling unit. The major drawback of this technique is that the entire application needs to be evaluated, either in detail or in fast-forward / warmup mode, and therefore the speedup is limited by the number of instructions in the whole application. There have been a number of application-specific multithreaded sampling methodologies (like BarrierPoint [**?**] and TaskPoint [**?**]) that move beyond some of the limitations of time-based sampling, speeding up simulation by considering the number of representatives ($O(rep)$), not the number of instructions ($O(insn)$), improving performance significantly.

The goal of our proposal is to speed up the generic multithreaded sampling by selecting the representative simulation regions from the whole application. Therefore, the performance of the whole application can be extrapolated from the performance of these representative samples.

2. Related Work

Table 1 shows the relevant prior work related to sampling methodologies. We can infer from the table that none of the previously proposed sampled simulation methodologies address the generic multi-threaded sampling problem that is independent of the length of the application.

3. Motivation and Scope

Sampling is a widely used technique to speed up simulation. The overall performance of an application can be extrapolated from the small number of instructions that are simulated in detail. Simpoint, SMARTS, and many other sampling methodologies are proposed for single-threaded applications, while multithreaded sampling remained an open problem until time-based sampling was proposed that uses time as the sampling unit. Time-based sampling simulation methodologies are slow as they need to simulate the whole application.

Sampling generic (synchronizing) multithreaded applications is tricky as instructions-per-cycle (IPC) is not a valid metric for such applications [**?**]. The forward progress of each thread is dependent on other threads as well due to the presence of synchronization primitives. The underlying hardware complexities contribute to the imbalance in thread progress. Hence, a generic multithreaded sampling methodology is essential to evaluate future hardware efficiently and would drive the computer architect to arrive at a design decision in no time.

4. Synchronization Agnostic Multithreaded Sampling

We consider a multithreaded application to be homogeneous if the workload of all the threads is balanced. We also take into account the underlying hardware that affects the execution behavior of an application. Hardware heterogeneity can be due to the presence of non-uniform memory access (NUMA) nodes, simultaneous multithreading (SMT), heterogeneous cores, etc. Figure 1 shows the overall workflow of the proposed methodology. The proposed methodology supports sampling homogeneous applications and statically scheduled heterogeneous applications running on homogeneous or heterogeneous hardware.

On homogeneous hardware, the application threads are assumed to progress equally, whereas on heterogeneous hardware, the threads progress differently. In order to build the correct thread ordering on heterogeneous hardware, we record

Type	Sampling methodology	Speed	Simulation time	Profiling required	Warmup	Comments
Single-threaded	Simpoint [**?**]	+++	O(rep)	Yes	Yes	
Single-threaded	SMARTS [**?**]	+	O(insn)	No	Yes	
Single-threaded	LiveSim [**?**]	++	O(rep)	Yes	No	Selects early representatives
Single-threaded	CoolSim [**?**]	++	O(insn)	No	No	Cache model
Single-threaded	Delorean [**?**]	+++	O(insn)	No	No	Cache model
Multithreaded	TurboSMARTS [**?**]	+	O(insn)	No	Yes	Non-synchronizing threads
Multithreaded	Time-based [**?**] [**?**]	+	O(insn)	Yes	No	
Multithreaded	BarrierPoints [**?**]	+++	O(rep)	Yes	Yes	Requires barriers
Multithreaded	TaskPoint [**?**]	+++	O(rep)	Yes	Yes	Requires tasks
Multithreaded	Proposed work	++	O(insn)	Yes	No	Requires static scheduled tasks

Table 1: Table shows various single-threaded and multithreaded application sampling techniques in-place. The field 'speed' is the amount of speedup that can be achieved using the corresponding sampling methodology when compared to the whole application simulation. The sampled simulation time depends on $O(rep)$ or on $O(insn)$.

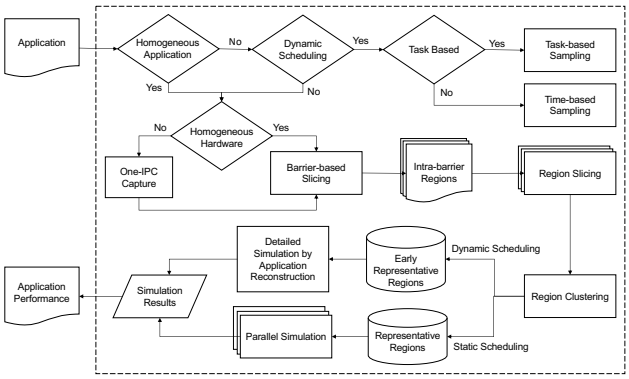

Figure 1: The overall flow of the proposed methodology.

the application execution using a one-IPC capture (all threads progress at the rate of one instruction per cycle) so that any effects of hardware heterogeneity on the application execution can be filtered out. The recorded execution path of the application is chopped into intra-barrier regions using a barrier-based sampling methodology like BarrierPoint [**?**]. Barriers are synchronization points in an application where no thread can make forward progress until all threads reach that point.

We split each intra-barrier region into slices of length 100 million instructions per thread. A global basic-block vector (BBV) is emitted when the instruction count reaches 100 million for at least one thread. The global BBV contains the basic-block information of all the threads.

The BBV emission is modified slightly in the presence of spinloops in the application. By providing an instruction target, a region can be chosen by looking for the next loop entry once the instruction target is achieved. The start and end of each region are described as an ordered-pair (PC, count), where 'PC' is the address of the corresponding region boundary marker and 'count' is the execution count of the marker at the start/end of the region. The value of count for a particular region size is invariant across multiple executions, which represents the unit of work done. This hence would remain as valid simulation points even in the presence of spin-loops.

The global BBVs generated from all such regions resulting from intra-barrier regions are clustered. The clustering phase is similar to Simpoint, except that we propose to use a hybrid clustering algorithm. We first use the k-means algorithm [**?**] to cluster the global BBVs and then use DBSCAN [**?**], keeping minimum samples per cluster to be one on each output cluster of the k-means algorithm. This aids in identifying heterogeneous regions in an application, and the resulting number of clusters represents the different phases that exist in the application.

For a homogeneous application, the region that lies the closest to the geometric centroid of the cluster (we use k-means to form clusters that have a centroid) is chosen as the representative of the cluster. The chosen representative samples are called simulation points. All representatives are simulated in parallel with ample warmup so that the microarchitectural state remains intact when the regions start simulating. In the case of a heterogeneous application, for each cluster, the earliest occurring region in the application is chosen to be the representative of that cluster. Similar to LiveSim, the simulation starts off with the first simulation point, and as the simulation moves further, the gaps are filled with similar regions executed previously and with previously recorded memory checkpoints.

5. Evaluation Methodology

We aim to modify Sniper multicore simulation infrastructure [**?**] to implement the proposed simulation methodology. We like to use the speed version of OpenMP applications in SPEC CPU 2017 benchmark suite [**?**] with eight threads. We plan to leverage Intel's Pin [**?**] and Pinplay [**?**] tools to generate reproducible, constrained, multi-threaded execution snapshots, called Pinballs [**?**], to allow for repeatable analysis. We will record the execution path of all applications as Pinballs. We will develop Pinplay-based tools to slice the whole application generating BBVs (of desired slice-size) and other vectors based on which the regions are clustered.

6. Conclusion

Sampling has been an effective technique employed by computer architects to reduce the simulation time from weeks or months to days or hours. We propose a generic multithreaded sampling methodology for applications running on heterogeneous hardware. The work aims to reduce the simulation time of generic multithreaded applications by orders of magnitude, which was not possible with prior methodologies. The essence of the proposed methodology lies in making full use of the phase behavior of the application.

References

[1] Daniel Sanchez and Christos Kozyrakis. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, page 475–486, New York, NY, USA, 2013. Association for Computing Machinery.

[2] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2011.

[3] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS X, page 45–57, New York, NY, USA, 2002. Association for Computing Machinery.

[4] Roland E. Wunderlich, Thomas F. Wenisch, Babak Falsafi, and James C. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, ISCA '03, pages 84–97, New York, NY, USA, 2003. ACM.

[5] T. E. Carlson, W. Heirman, and L. Eeckhout. Sampled simulation of multi-threaded applications. In *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–12, April 2013.

[6] E. K. Ardestani and J. Renau. Esesc: A fast multicore simulator using time-based sampling. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pages 448–459, Feb 2013.

[7] T. E. Carlson, W. Heirman, K. Van Craeynest, and L. Eeckhout. Barrierpoint: Sampled simulation of multi-threaded applications. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–12, March 2014.

[8] T. Grass, T. E. Carlson, A. Rico, G. Ceballos, E. Ayguadé, M. Casas, and M. Moreto. Sampled simulation of task-based programs. *IEEE Transactions on Computers*, 68(2):255–269, Feb 2019.

[9] S. Hassani, G. Southern, and J. Renau. Livesim: Going live with microarchitecture simulation. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 606–617, March 2016.

[10] N. Nikoleris, A. Sandberg, E. Hagersten, and T. E. Carlson. Coolsim: Statistical techniques to replace cache warming with efficient, virtualized profiling. In *2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS)*, pages 106–115, July 2016.

[11] Nikos Nikoleris, Lieven Eeckhout, Erik Hagersten, and Trevor E. Carlson. Directed statistical warming through time traveling. In *Proceedings of the 52Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, pages 1037–1049, New York, NY, USA, 2019. ACM.

[12] Thomas F. Wenisch, Roland E. Wunderlich, Babak Falsafi, and James C. Hoe. TurboSMARTS: Accurate microarchitecture simulation sampling in minutes. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '05, pages 408–409, New York, NY, USA, 2005. ACM.

[13] T. Grass, A. Rico, M. Casas, M. Moreto, and E. Ayguadé. Taskpoint: Sampled simulation of task-based programs. In *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 296–306, April 2016.

[14] A. R. Alameldeen and D. A. Wood. Ipc considered harmful for multiprocessor workloads. *IEEE Micro*, 26(4):8–17, July 2006.

[15] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[16] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[17] James Bucek, Klaus-Dieter Lange, and Jóakim v. Kistowski. Spec cpu2017: Next-generation compute benchmark. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, ICPE '18, pages 41–42, New York, NY, USA, 2018. ACM.

[18] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '05, pages 190–200, New York, NY, USA, 2005. ACM.

[19] Harish Patil, Cristiano Pereira, Mack Stallcup, Gregory Lueck, and James Cownie. Pinplay: A framework for deterministic replay and reproducible analysis of parallel programs. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '10, pages 2–11, New York, NY, USA, 2010. ACM.

[20] Harish Patil and Trevor Carlson. Pinballs: portable and shareable user-level checkpoints for reproducible analysis and simulation. In *REPRODUCE : Proceedings of the Workshop on Reproducible Research Methodologies, Abstracts*, page 2, 2014.