Using SimPoint for Accurate and Efficient Simulation

Erez Perelman Greg Hamerly Michael Van Biesbrouck
Timothy Sherwood Brad Calder

Department of Computer Science and Engineering
University of California, San Diego
{eperelma,ghamerly,mvanbies,sherwood,calder}@cs.ucsd.edu

Abstract

Modern architecture research relies heavily on detailed pipeline simulation. Simulating the full execution of a single industry standard benchmark at this level of detail takes on the order of months to complete. This problem is exacerbated by the fact that to properly perform an architectural evaluation requires multiple benchmarks to be evaluated across many separate runs. To address this issue we recently created a tool called SimPoint that automatically finds a small set of Simulation Points to represent the complete execution of a program for efficient and accurate simulation. In this paper we describe how to use the SimPoint tool, and introduce an improved SimPoint algorithm designed to significantly reduce the simulation time required when the simulation environment relies upon fast-forwarding.

Keywords
SimPoint, Clustering, Simulation, Fast-forwarding, Sampling

1. SIMPOINT

Understanding the cycle level behavior of a processor running an application is crucial to modern computer architecture research. To gain this understanding, detailed cycle level simulators are typically employed. Unfortunately, this level of detail comes at the cost of speed, and simulating the full execution of an industry standard benchmark on even the fastest simulator can take weeks to months to complete. This fact has not gone unnoticed in the academic community, and several researchers have started to develop techniques aimed at reducing simulation time.

For architecture research it is often necessary to take one instance of a program with a given input, and simulate its performance over many different architecture configurations. The same program binary with the input may be run hundreds or thousands of times to examine how, for example, the effectiveness of a given architecture changes with its size. Our goal in creating SimPoint [1, 2] is to (1) significantly reduce simulation time, (2) provide an accurate characterization of the full program, and (3) to perform the analysis to accomplish the first two goals in a matter of minutes. These goals are met by simulating only a handful of *intelligently* chosen sections of the full program. When these sections (simulation points) are carefully chosen, it provides an accurate picture of the complete execution of the program and results in highly accurate estimations of performance

The key to our approach is that for a given program and input, the simulation points only need to be chosen once. This is because we select them using a method that is independent of any particular architecture configuration. The simulation points are selected using a metric that is only based on the code that is executed over time for a program/input pair. Once the simulation points are chosen they can be used for the hundreds or thousands of independent simulations that may be needed, significantly reducing simulation time.

To pick the simulation points in [1, 2], we introduce the concept of profiling *Basic Block Vectors* (BBV) as a way of capturing the important behaviors of the program over time [1]. A Basic Block Vector captures the relative frequency of the code blocks executed during a given portion of execution. After profiling a program with a particular input, we compare the basic block vectors to see how similar they are to one another. Intervals of execution that execute the same code blocks with the same frequency are grouped together into clusters using clustering algorithms from machine learning. We found that sections of execution (represented by basic block vectors) that are grouped into the same cluster have very similar behavior across all the architecture metrics we have examined. Once we break the program into clusters, we pick a single point from each cluster (appropriately weighted) to serve as its representative. The set of representative sections are where detailed simulation is performed. Simulating only these simulation points provides an accurate and efficient representation of the complete execution of the program. All of the code to track the basic blocks, perform the analysis, do the clustering, and pick the simulation points is distributed as part of our SimPoint tool.

1.1 Using SimPoint for Simulation

A run of the SimPoint tool begins by running each program and input pair through the basic block tracker to generate the basic block vectors. Currently we support gathering the vectors with either ATOM or SimpleScalar. These vectors are then run through the analysis portion of the tool, which does the clustering and generates the points for simulation. One input parameter worth mentioning is the granularity, or interval size. The interval size determines how large of a chuck (in instructions) the program's execution should be divided into. Therefore, it determines the length of simulation that will be required for any simulation point used. In this paper we use a granularity of 10 million instructions, whereas our prior work [2] used 100 million instructions per simulation point. In addition to setting the granularity, it is also possible to set the ceiling on the maximum number of samples (clusters) you are willing to simulate, and SimPoint will find the most representative simulation points under those constraints. The output from our tool is a set of points to be simulated and their corresponding weights (based on the percent of execution that a given simulation point represents). This is used to weight the corresponding metrics sampled for a simulation point, when combining the results together to get an overall metric for simulating the program/input pair.

When using simulation points or sampling, the issue of how to warmup the architecture structures needs to be dealt with. The three approaches we have examined are using checkpoints, stale state, and no warmup at all.

If the simulation environment supports checkpoints, then a checkpoint can be made at the start of each simulation point.

SIGMETRICS'03, June 10–14, 2003, San Diego, California, USA.
ACM 1-58113-664-1/03/0006.

This avoids fast-forwarding to each simulation point for each run of the program/input. In addition, all of the simulation points can even be run in parallel to obtain extremely fast results for one particular program/input pair.

If the simulation environment does not support check-pointing, then the simulation must fast-forward between the simulation points. When large simulation intervals are used (e.g., 100 million instructions) we found that no warmup is needed, since any cold-start effects are insignificant in comparison to the effect of execution. If no warmup is used, all of the large architecture structures (e.g., cache, branch predictors) make use of a warmup bit that indicates when the first time an entry is used. If it is the first time, the access is assumed to be a hit or a correct prediction. This a very simple method we added to SimpleScalar, and provides fairly accurate warmup state, since the miss rates for these structures are usually fairly low. For smaller interval sizes (e.g., 10 million) we found that it is beneficial to use *Stale State* or other proposed warmup techniques to reduce bias from cold-start effects. Stale state is a method of not resetting the architecture structures between simulation points, and instead uses them in the state they were in at the end of the prior simulation point we just fast-forwarded from.

1.2 Early Simulation Points

In [2], the goal was to pick a single simulation point from each cluster that best represents all the intervals of execution in that phase. For simulation environments that do not support check-pointing, it can require up to several days to fast-forward to the latter part of the execution if that is where the simulation point is located. Our goal is to find early simulation points to significantly reduce time spent fast-forwarding while still accurately representing the overall execution of the program.

If we consider a simulation environment where multiple points can be simulated one after the other (a single run is done through the program and detailed simulation is interleaved with periods of fast-forwarding), then the *last* simulation point in program execution order will determine the simulation time. Our Early SimPoint algorithm focuses on choosing a clustering that is both representative of the program's execution and has some feasible simulation points early in the program for all clusters (if possible). This might not be achievable for all program's, since an important phase of execution may truly only appear at the end of the program's execution. We therefore give priority in our algorithm towards still making sure that the clustering represents the overall execution of the program. Once the early clustering is performed, we choose a representative simulation point early in the execution from each cluster.

Figure 1 shows the percent error in IPC using simulation points from [2], and Early simulation points. The error is calculated by comparing the estimated error using the different sampling techniques, to the IPC found doing detailed simulation of all the programs to completion. The simulation points from [2] assume perfect warmup, and are collected with interval sizes of 100 million. For the Early results, we used an interval size of 10 million, and each program had less than 30 simulation points. The early simulation points use stale state as described above to reduce bias from cold-start effects. The results show that the early simulation points have a slightly higher error rate, 3.5% on average, over the 2.1% error from the original simulation points from [2].

In terms of simulation time, Figure 2 shows the number of instructions required to fast-forward for these simulation techniques. Here the benefit of using early simulation points over the former simulation points is clearly seen. The results show

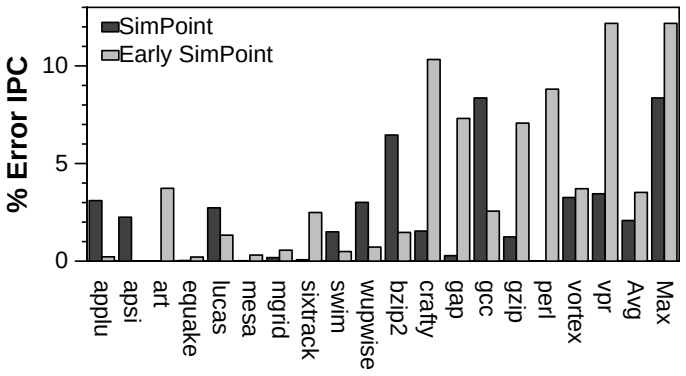

Figure 1: IPC relative error for the Original SimPoint and Early SimPoint algorithms.

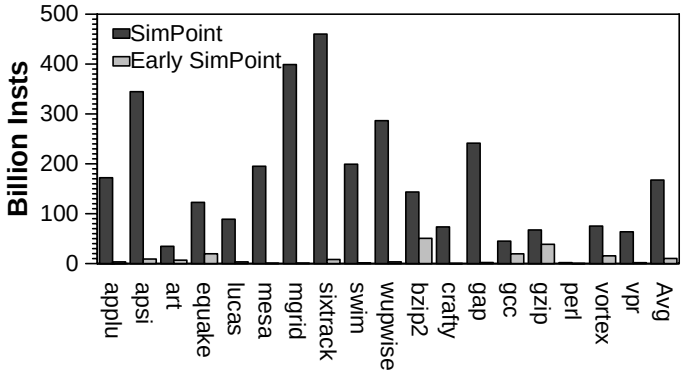

Figure 2: Number of instructions to fast-forward for simulation.

that using early simulation points reduces simulation time more than 15 times over the prior SimPoint algorithm in [2].

Summary: Even on the fastest cycle level simulators, simulating a single minute of real time can take 50 days. Even doing very simple emulation of a program for the purpose of fast-forwarding can take on the order of days. SimPoint is a powerful, available, and easy to use tool for attacking this problem. It requires little or no modification of most cycle level simulators used today, and only needs to be run once per program input pair to pick the simulation points. SimPoint will perform a fully automated analysis of a profile to determine where to best spend limited simulation program analysis resources, and our new Early SimPoint technique makes the approach even faster on simulators without support for check-pointing. Using SimPoint we have found errors always to be less than 8.4%, and even with Early SimPoint the error does not go above 12.2%. Early SimPoint has the benefit that the average fast-forwarding amount is reduced from 167 billion instructions to less than 11 billion instruction.

This work was funded in part by NSF CAREER grant No. CCR-9733278, Semiconductor Research Corporation grant No. SRC-2001-HJ-897, and an equipment grant from Intel.

2. REFERENCES

[1] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *International Conference on Parallel Architectures and Compilation Techniques*, September 2001.

[2] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *10th International Conference on Architectural Support for Programming*, October 2002.