On the Use of Trace Sampling for Architectural Studies of Desktop Applications

Patrick Crowley & Jean-Loup Baer
Department of Computer Science & Engineering
University of Washington
Seattle, WA
{pcrowley, baer}@cs.washington.edu

1. INTRODUCTION

Trace-driven simulation is a common approach for evaluating memory systems. Unfortunately, it also demands large amounts of space and time, particularly for large caches and long running applications. These demands can be greatly reduced by employing sampling techniques at the expense of providing only a statistical estimate of the properties of a full trace.

Our interest in using sampling is three-fold. First, we are interested in the behavior of commonly used desktop applications. When compared to benchmarks such as SPEC95, these applications have larger working sets, are feature rich, and, of course, can run for billions of instructions. Hence, traces based on exhaustive executions of such applications will be large. In this work, we consider trace sampling for a suite of five publicly available desktop application traces for Windows NT on the Intel X86 platform. To determine the feasibility of sampling for these traces, we present cache miss rate results for a large set of cache sizes and sampling techniques. Second, we want to demonstrate the utility of these sampling techniques for architectural studies. Although it has been shown that trace sampling is not very accurate for metrics such as hit rate when simulating large multi-megabyte caches[3], we demonstrate that sampling is useful in assessing trends not only for caches but also for other architectural structures whose state depends on the processing of past references. Two architectural studies are presented here that apply sampling techniques. The first study demonstrates how sampling may be used to assess trends in victim cache performance[2], and the second study uses sampling to assess trends in branch prediction techniques. In addition, we can estimate parameters for analytical cache models with sampled traces as precisely as estimating with whole reference traces (cf. [1] for this part of the study).

2. TRACE SAMPLING

In trace sampling, an observation, or *sample*, is obtained by recording a fixed number, the *sample size*, of consecutive references from a reference stream. Another fixed number of references are ignored before the next observation is made. The *sampling ratio* is the percentage of total references used in all the observations.

Sampling theory states that sets of random, unbiased observations from a population may be used to make inferences about that population. As described above, observations in trace sampling are not random; they are systematic since they are evenly spaced throughout the trace. This non-random pattern is not a problem, though, since systematic observations can be used to make even more precise inferences than random observations when the variance of systematic observations is greater than the variance of the population. Unfortunately, however, trace sampling for memory systems neither involves unbiased observations nor a sufficient alternative. The problem is that the state of the cache is unknown at the start of each observation, i.e., within a sample it is unknown whether the first reference to each cache block will be a hit or a miss. These are known as *unknown*[7] or *cold-start*[4] references.

3. METHODOLOGY

3.1 Sampling Techniques

A number of techniques have been employed to mitigate the bias due to unknown references. The techniques considered here are described in Figure 1. One approach is to make assumptions about, or construct, the state of the cache at the start of each sample (e.g., **cold**, **half**, and **stitch**). Another is to directly estimate the miss ratio of unknown references (e.g., **INITMR**). The efficacy of these assumptions depends on workload, cache organization, and choice of sampling parameters. For example, if most misses in a cache for a given workload are due to conflicts in a small number of cache lines, then **stitch** may work well since only a small portion of the working set is likely to change between samples.

Technique	*Description*
cold	each unknown reference misses
half	first half of each sample used to prime the cache[4]
stitch	uses the end state of the previous sample as the start state for the current sample
INITMR	estimates the miss ratio of unknown references[3]
true-sample	Starts each sample with the correct cache state

Figure 1 . Sampling Techniques

Note that **true-sample** simulates the caches over the full trace and reports the miss ratio observed over the regions that are sampled with the other techniques. It is therefore an unbiased estimator of the miss ratio for the entire trace. Its accuracy depends on how ''fine-tuned" our sampling parameters, sample size and sampling ratio, are to a given cache and workload. While **true-sample** is not a practical method, it is however the basis for comparisons with the other techniques which, in addition to the same sampling errors, will have unknown reference biases.

3.2 Benchmarks

Our experiments are based on traces gathered from the execution of five desktop applications: Adobe Acrobat Reader, Netscape Navigator, Adobe PhotoShop, Microsoft PowerPoint, and Microsoft Word. A complete description of these applications and their workloads can be found in [5].

4. SAMPLING RESULTS

4.1 Determination of Miss Rates

We simulated direct mapped and 4-way set-associative instruction and data caches with sizes ranging from 8KB to 128KB and direct-mapped and 4-way set-associative combined caches with sizes ranging from 256KB to 4MB. We include 90% confidence intervals for all sampling results to give each estimate context. The following observations are representative of the complete set of simulations.

1. The real miss rate is within the 90% confidence interval of **true-sample** for all experiments.
2. All techniques are accurate within the 90% confidence interval for caches up to 32KB in size.
3. **stitch** and **INITMR** are reliable for these workloads up to 64KB, and **stitch** is the most accurate.
4. No technique is reliable for all large caches, although **stitch** comes quite close.

4.2 Architectural Studies

Trace-driven simulation is used not only for the evaluation of cache parameters but also for studying hardware assists to the memory hierarchy or to the processor core. Often these hardware assists contain structures which, like caches, have states that depend on the recent history of data references or instruction execution.

4.2.1 Victim Caches

The scenario we consider here is that of an architect who wishes to gather an efficient estimate of the expected change in miss rate for data caches augmented with victim caches of between one and five entries. Our experiments compare the true results for the victim cache simulations to **stitch** sampled results, whose miss rate accuracy has already been discussed. In addition to the actual miss rates being very similar, the true miss rate is always within the 90% confidence interval (or just outside) for caches up to 128KB in size. The trends in improved miss rate due to the victim caches are correctly predicted, even for large caches.

4.2.2 Branch Prediction

Branch prediction is typically simulated over the same workloads used to drive cache memory simulations. As these prediction techniques increase in complexity, so do the costs of simulation. Our experiments compare the complete branch prediction accuracy results to **stitch** sampled results for two branch predictors: a simple bimodal[8] predictor and a gshare[6] predictor. We consider predictor tables ranging in size from 512 to 32,768 entries.

The results using **stitch** for the bimodal predictor are extremely good, even for large tables. Because individual branches are most often either always taken or always not taken, keeping the results of past predictions, even distant ones, is beneficial. Regarding the gshare predictor, we observe that **stitch** is still quite accurate up to 8K entries. Moreover, **stitch** can lead to the correct conclusion that the gshare predictor is more accurate than the bimodal predictor only when the predictor tables are sufficiently large.

5. CONCLUSIONS

Commonly used Windows NT desktop applications can run for billions of instructions. Performing architectural studies on traces of billions of references is not feasible from both time and space perspectives. Trace sampling is an efficient alternative that greatly reduces simulation time and, in many cases, nominally reduces accuracy.

We have shown that among the choices of sampling techniques, **stitch** most accurately predicts cache miss rates for these workloads. Furthermore, we have shown that sampling can be used to efficiently drive architectural studies and gather parameters for analytical cache models.

6. ACKNOWLEDGEMENTS

This work was supported in part by NSF Grant MIP-9700970 and by a gift from Intel Corporation.

7. REFERENCES

[1] P.J. Crowley and J-L Baer. On the use of trace sampling for architectural studies of desktop applications. Technical Report UW-CSE-98-12-05, University of Washington, 1998.

[2] N. Jouppi. Improving direct-mapped cache peformance by the addition of a small fully-associative cache and prefetch buffers. In *Proc. of 17th Int. Symp. on Comp. Architecture*.

[3] R.E. Kessler, et al. A comparison of trace-sampling techniques for multi-megabyte caches. *IEEE Trans. on Computers*, 43(6):664-675, June 1994.

[4] S. Laha, et al. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Trans. on Computers*, 37(11):1325-1335, Nov. 1988.

[5] D. C. Lee, et al. Execution characteristics of desktop applications on Windows NT. *In Proc. of 25th Int. Symp. on Comp. Architecture*, June 1998.

[6] Scott McFarling. Combining Branch Predictors. Technical Report TN 36, DEC-WRL, 1993.

[7] D. A. Wood, et al. A model for estimating trace-sample miss ratios. In *Proc. of ACM SIGMETRICS Conf. for the Measurement and Modeling of Comp. Systems*, pp. 79-89, June 1991.

[8] T.-H. Yeh and Y. Patt. Alternative implementations of two-level adaptive branch prediction. *In Proc. of 19th Int. Symp. on Comp. Architecture*, pp. 124-134, 1992.