Combining Cluster Sampling with Single Pass Methods for Efficient Sampling Regimen Design

Paul D. Bryan Thomas M. Conte
Center for Efficient, Secure and Reliable Computing (CESR)
North Carolina State University, Raleigh, NC 27695
{pdbryan, conte}@ncsu.edu

Abstract

Microarchitectural simulation is orders of magnitude slower than native execution. As more elements are accurately modeled, problems associated with slow simulation are further exacerbated. Given these issues, many researchers have devised sampling techniques to reduce simulation time.

When cluster sampling techniques are used, care must be taken to remove sampling and non-sampling biases. Researchers have devised clever methods for effectively reducing non-sampling bias, but little work has been proposed for efficient reduction of sampling bias (sampling regimen design).

Traditionally, sampling regimen design has been an iterative process that required a full workload simulation for error comparison. In this study, a single-pass simulation technique for sampling regimen design is proposed. Using this method, thousands of sampling regimen candidates can be simultaneously evaluated. With this technique, simulation speed was increased by an average factor of 17 with a maximum increase of 73 times relative to the total workload simulation. Additionally, this technique allows the user to effectively estimate the sample error without running the entire workload.

1. Introduction

Modern processor design is driven by simulation. Architects propose new design features and then simulate to determine the efficacy of their designs. In order to make valid inferences from a simulation, often long instruction traces must be simulated. Furthermore, the instruction counts of contemporary benchmark suites are exploding to become orders of magnitude larger than their previous counterparts (e.g., spec2006 vs. spec2000). To compound this issue, the process of simulation is increasingly slower as more cycle-accurate features are modeled. A program that can be executed fully on hardware in a matter of minutes, can take weeks or months to simulate. Given that simulation is a limiting factor to new processor technologies, many researchers have devised methods to reduce simulation time.

Historically, researchers often executed an arbitrary instruction stream located after initialization code in a benchmark. Although effective in reducing simulation time, the arbitrary selection of instructions can lead to inferences that are misleading or inaccurate. Currently, researchers often rely on sampling techniques to reduce the number of instructions required for simulation. Two commonly used approaches include SMARTS [20] and SimPoint [18].

A number of sampling techniques have been applied to hardware simulation, which include *cluster sampling* [2],[5],[12],[17], *set sampling* [7],[10],[12], and *stratified sampling* [14]. These methods differ in which elements are sampled from the overall population, but they all strive to extract a subset of elements in order to reduce simulation time. In cluster sampling, a group of contiguous elements from the population is selected to form a cluster. The information obtained from the cluster is then used for measurement as an individual sampling unit.

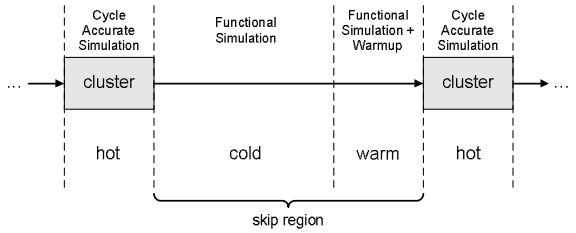

Figure 1: Cluster Sampling

Figure 1 shows the general representation of cluster sampling when applied to processor simulation. The horizontal line represents the entire dynamic instruction stream for the workload of interest. From the instruction stream, clusters of instructions are selected. These clusters are the regions where cycle-accurate measurements are taken. The figure shows three regions of execution: hot, cold, and warm. Hot simulation refers to the complete cycle-accurate simulation of the system. The pipeline, memory hierarchy, branch predictor, etc., are all simulated

within the hot phase. Generally, hot execution consists of normal system simulation. Once the cluster has finished, execution continues in the cold phase. The cold phase consists of simple functional simulation. The purpose of cold simulation is to ensure correct architectural and functional memory state. At a predetermined point prior to the next cluster, the warm execution phase begins. In warm execution, data are functionally applied to high-state microarchitectural elements, such as the branch predictor and memory hierarchy. Functional simulation continues identically in the cold phase, but the elements are not rigorously modeled as in the hot phase. The purpose of warm execution is to warm-up the state of the processor before measurements are taken from the next cluster.

When performing any type of sampling scheme, two types of biases must first be overcome: sampling bias and non-sampling bias. Sampling bias refers to the construction of a representative sample of the overall population. For processor simulation, sampling bias is reduced with the design of a representative sampling regimen. Non-sampling bias is formally described as all other bias that is not sampling bias. For processor simulation, non-sampling bias is the difference in processor state that causes dissimilar measurement from the full workload simulation. The two most important structures pertaining to non-sampling bias are the memory hierarchy and branch predictor.

Non-sampling bias is removed through the use of warm-up methods. Previously, warm-up methods have been the main focus of sampling techniques. Many algorithms including MRRL [8], BLRL [6], and SMARTS [20] have been extensively studied. However, the design of a valid sampling regimen is often overlooked. Most authors simply state the sampling regimen they used but neglect showing how it was derived.

This paper focuses on the development of a method to efficiently determine a valid sampling regimen that effectively reduces sampling bias. The techniques presented in this paper allow a user to generate a sampling regimen without executing the entire workload for comparison. Moreover, the single pass nature of the algorithm alleviates the traditional iterative process to sampling regimen design.

2. Statistical Sampling Assumptions

According to the central limit theorem, randomly extracted data from any non-normal distribution generates a normal distribution. From this normal distribution, conventional associations and formulas may then be applied to the normal distribution to find its mean, variance, etc. The inferences drawn from the normal distribution may then be inferred to be associated with the non-normal distribution. When sampling from a large population of any distribution shape, the distribution of the sample means will approach the normal distribution with a sufficient sample size [9].

Two fundamental assumptions regarding sampling techniques are as follows: 1) increasing the sample size will increase the sample accuracy; and 2) the test for individual inclusion in the sample must be random, and each element in the population must have the same probability for inclusion. With these assumptions, confidence intervals can then be calculated as follows:

$$S_{IPC} = \sqrt{\frac{\sum_{i=1}^{N_{cluster}} (\mu_{IPC}^{i} - \mu_{IPC}^{sample})^2}{N_{cluster} - 1}}, \qquad S_{\overline{IPC}} = \frac{S_{IPC}}{\sqrt{N_{cluster}}},$$

S_{IPC} is the standard deviation and $S_{\overline{IPC}}$ is the standard error for a cluster sampling design. The estimated standard error is used to calculate the *error bounds* and *confidence interval*. Using the properties of the normal distribution, the 95% confidence interval is given by $\mu_{IPC}^{sample} \pm 1.96\ S_{IPC}$, where the error bound is $\pm 1.96 S_{IPC}$. A confidence interval of 95% implies that 95 out of 100 sample estimates may be expected to fit within this interval. Low standard errors imply relatively small variations in repeated estimates, and consequently, result in higher precision. For each sample, the relative error is calculated as follows:

$$RE(IPC) = \frac{\left| \mu_{IPC}^{true} - \mu_{IPC}^{sample} \right|}{\mu_{IPC}^{true}},$$

where μ_{IPC}^{true} is the true population mean (IPC), and μ_{IPC}^{sample} is the estimated mean (IPC) obtained from the sample. Relative error relies on μ_{IPC}^{true} from a full simulation of each workload.

A variety of factors affect sampling accuracy for a constant sample size, including the variance of the population and the type of distribution being sampled. However, as the sample size increases, accuracy also increases. Thus, with a sufficiently large sample size, the estimate of the mean approaches the true mean and the error approaches zero.

3. Sampling Regimen Construction

When designing a valid sampling regimen, there are three parameters that must be determined: the cluster size, the number of clusters, and the location of the clusters within the dynamic instruction stream. The cluster size refers to the number of instructions executed in full cycle-accurate detail. The cluster's number and size dictate the sample size. Statistical simulation is a compromise between speed and accuracy. If too many clusters are selected, then speed will be sacrificed. If too few clusters are selected, then accuracy will be sacrificed. To further complicate this issue, the location of clusters is equally important. Even if the cluster's number and size are sufficient for sampling performance, the clusters may be located at non-representative sections of code. Thus, measurements taken at such locations would lead to inaccurate estimates of performance.

Many researchers have investigated techniques in reducing the *cold-start* bias for microarchitectural simulation. However, little work has been proposed dealing with efficient techniques for sampling regimen design.

When designing a sampling regimen, each program-input pair may have dramatically different performance, affecting the underlying distribution of IPC. Thus, a sampling regimen that performs well for one workload will not necessarily be accurate when applied to other workloads (or even the same workload with different inputs).

Often, sampling regimen parameters are derived through an iterative process consisting of the following steps: 1) the entire workload is executed for error comparison; 2) the workload is sampled according to a predefined cluster number and size; 3) the results are analyzed to determine if confidence tests are met with sufficiently low error; and 4) if the error threshold or confidence tests are unsatisfactory, return to Step 2.

This iterative process can be extremely time-consuming. For example, consider a user who wishes to evaluate a number of sampling regimen configurations. Assume the user wishes to assess cluster sizes ranging from 1000 to 50000, with a step size of 1000, and a cluster number ranging between 30 and 1000. For simplicity, assume each execution takes 30 minutes. The total time required to evaluate the entire suite of regimen configurations would take over 48,000 simulations which would require over 2 years of processor time. Even more time would be necessary if multiple random seeds are considered for each sampling regimen. Although it is not likely anyone would ever perform such an exhaustive regimen sweep, the previous scenario represents an extreme example of how time-consuming regimen design can grow.

Although a sampling regimen may be valid for a given workload, the user may be unlucky and randomly select non-representative locations. For example, a user could simulate the same sampling regimen 20 times in a row (utilizing different random seeds) and statistically, 1 could fail (a 95% confidence test implies that the true population mean has a 95% chance of being bracketed by the sample estimate). Therefore, a valid sampling regimen could be evaluated as a "false negative."

The purpose of this work is to allow users to perform a single simulation to derive a valid sampling regimen and achieve the following goals: 1) prevent users from having to run the entire workload for performance comparison; 2) circumvent the iterative nature of regimen design; and 3) derive a valid sampling regimen according to user-specified criteria.

4. Single-Pass Regimen Design

Figure 2 shows a flow diagram of the single-pass regimen evaluation. First, a program-input pair is profiled via a large sample. This large sample contains a very large number of clusters, many times more than required for a valid sampling regimen. This sample is called the *over-sample* because the sampling rate is much higher than the minimum requirements. As the sample size increases, the estimate of the mean will converge to the true mean, and the over-sample should estimate the mean very accurately. Embedded in this sample, information is contained regarding varying sized clusters.

After the over-sample has been collected, the estimate of the mean, or IPC, is assumed to be a highly accurate estimate of the true performance of the workload. The data collected in the over-sample is then analyzed to generate a list of sampling regimen candidates. Given user-specified criteria, the candidate list is pruned. The final candidate list then identifies valid sampling regimen configurations. Reported data include the cluster size, cluster number, and starting location of each cluster.

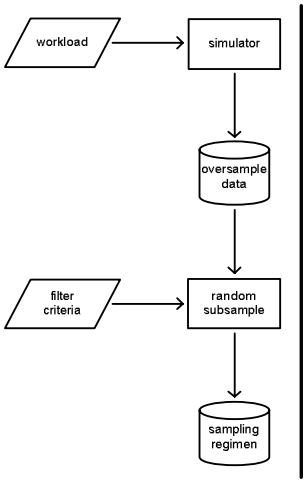

Figure 2: Single-Pass Sampling Regimen Flowchart

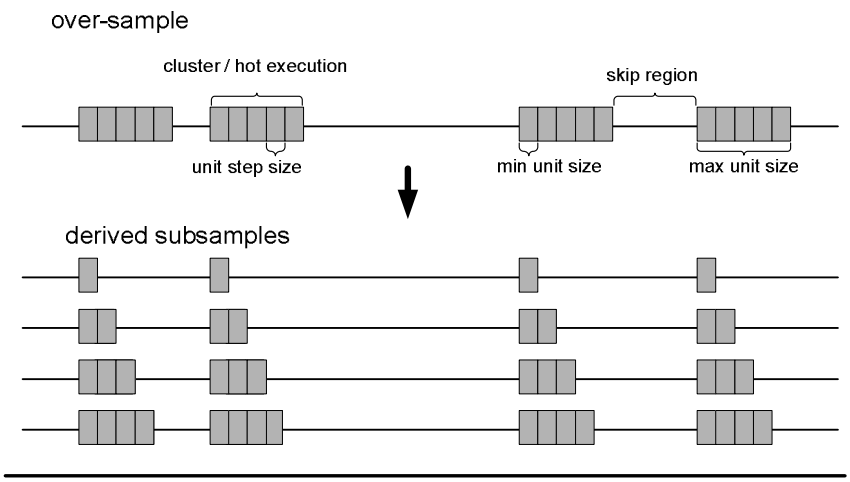

Figure 3: Cluster Sampling Modifications to Enable Derived Subsamples

4.1. Simulator Modifications

If a user has a simulator that implements cluster sampling with SMARTS [20], the modifications to implement single-pass regimen design are minimal. This work was performed using SMARTS warm-up since it is generally accepted as the most accurate warm-up method. However, this method is orthogonal to warm-up, and could be applied using any warm-up method that effectively removes non-sampling bias.

The simulator was modified to ensure all aspects of the regimen configuration could be controlled; in this paper a configuration file was used to specify the regimen attributes. Parameters could be specified via the command line, but the large number of inputs would make this approach unattractive. The regimen configuration file is used to specify the size of a cluster, the number of clusters, and the starting location of each cluster according to the dynamic instruction stream count. Additional parameters for the profile sample include a minimum cluster size, a maximum cluster size, and a step size.

Execution of each cluster continues similarly as SMARTS with a few minor differences. As in SMARTS, warm-up is performed on structures such as the branch predictor and memory hierarchy as instructions are skipped. When a cluster is encountered, normal execution continues where each cluster is evaluated according to the specified minimum size. The difference between single-pass and SMARTS is hot execution continues past the minimum cluster size, until the maximum cluster size has been reached. At each step size increment, all measured information in a cluster is checkpointed. The checkpointed information is recorded to allow the user to determine which measurements would have been taken for each cluster size and location. Figure 3 shows a diagram of the accounting differences between normal SMARTS execution.

4.2. Profiling Sample

Figures 4, 5, and 6 show the results of the profile simulations. As expected, increasing the sample size increases accuracy, but also increases simulation time.

Each workload was sampled such that a specified ratio of the entire workload was included in the over-sample simulation. Each workload was sampled according to the following sampled to non-sampled ratios: 1:6, 1:12, 1:24, 1:48, 1:96, 1:192, 1:384, 1:768, 1:1536, and 1:3072. The 1:6 sample ratio indicates one out of every six instructions in the overall workload execution was simulated in full detail.

Figure 4 shows the sampling accuracy associated with each sampling ratio. As expected, the smallest sampling ratio, 1:3072, (corresponding to 0.03%) had the highest error. For this ratio, *gcc* had the highest error with 24%, and *art* had the lowest error with 0.4%. As the sampling ratios increased, the error rates for all benchmarks decreased. At the highest sampling ratio, the average relative error was 0.03%. Figure 5 shows a magnified version of Figure 4 at the largest two sample sizes. As shown, a sampling ratio of 1:6 achieved an average relative error of 0.3%. At

this sampling ratio, *ammp* had the highest relative error at 1.7%.

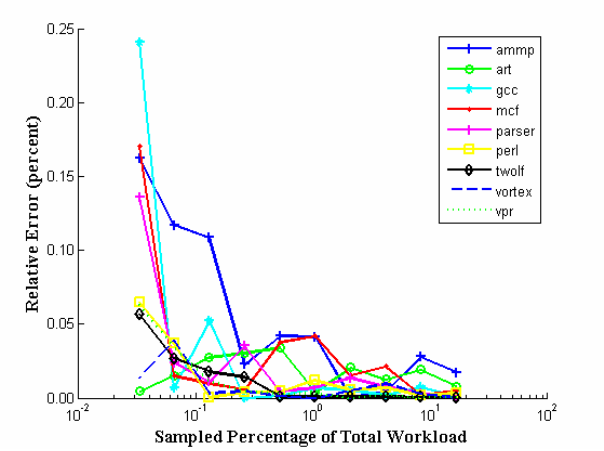

Figure 4

As shown in Figure 6, the time required for simulation explodes for certain benchmarks. At the highest sampling ratio, 1:6, the average execution time was approximately 5.4 hours. *Mcf* took the longest at this ratio with 22.8 hours. However, the largest sampled ratio is not necessary for accurate estimation. Very similar accuracy was obtained with a sampling ratio of 1:48. At this sampling ratio, the average relative error was 0.7% with an average execution time of 1.5 hours.

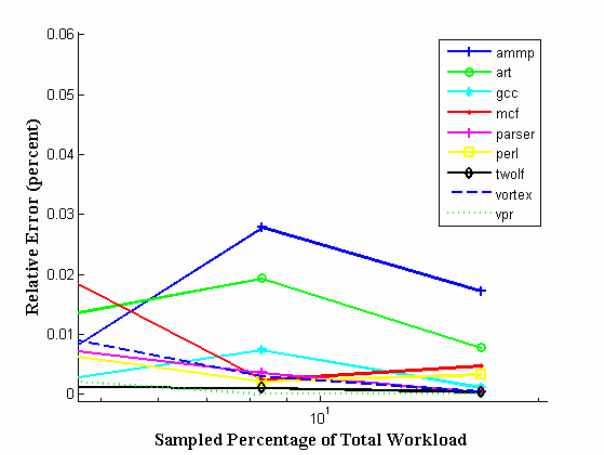

Figure 5

Although accuracy converges at a different sampling ratio for each workload, all experiments obtained accurate results when a sampling ratio of 1:48 was used.

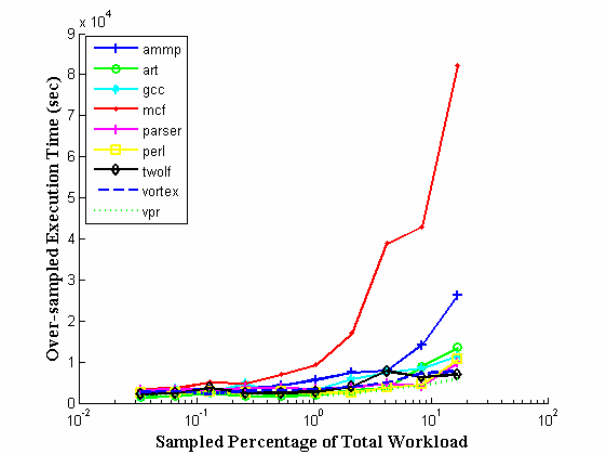

Figure 6

4.3. Profiling Analysis

Once the over-sampled data have been collected, the results must be analyzed. Using this information, random subsamples are compiled over all ranges of possible cluster sizes and numbers of clusters. For each cluster size and cluster number, elements are extracted randomly from the over-sampled population. Each over-sampled subset is then evaluated in terms of relative error, variance, and statistical confidence.

As previously stated, it is possible for a particular sampling regimen configuration to be rejected because non-representative elements from the population were included in the sample. To counter this possibility, each sampling regimen configuration is tested multiple times. From these results, the average expected performance for a given sampling regimen can be obtained. Each sampling regimen configuration is then evaluated multiple times with different random seeds. In this work, each sampling regimen was tested 30 times in correspondence with the central limit theorem. Once all possible input combinations of cluster sizes and the number of clusters have been searched, candidate selection proceeds.

The benefit of this profile analysis is twofold. First, the total workload simulation is not required since the over-sampled population mean is assumed to have sufficiently converged on the true population mean. Second, all cluster sizes and numbers of clusters recorded in the over-sampled population can be simultaneously evaluated for inclusion.

The methods detailed in the candidate analysis are statistically valid since each element in the over-sampled population has the same probability of being selected. Furthermore, use of the over-sampled data

provide confidence bounds that bracket the true performance of each workload.

4.4. Candidate Selection

Once the profile sample has been analyzed and a list of candidates generated, a sampling regimen configuration must be selected. Using the candidate list, candidates are pruned according to user-specified criteria. In this study, elements from the candidate list were excluded if all trials did not pass confidence tests relative to the over-sampled estimate. The minimum error could not exceed 2% and the variance could not fall below 0.02. These threshold values are somewhat arbitrary, and can be tuned to whatever characteristics the user desires.

Some error is expected to be present when comparing the full execution of a workload with the over-sample. Additionally, some error is expected to be present when comparing the second level simulation, provided by candidate selection, with the over-sample. By restricting the minimum error to 2%, the total bias introduced in the sampling process should be kept sufficiently low to bracket the true performance of the workload. Since variance is proportional to the confidence interval bounds, a higher variance increases the probability that the true performance will be bracketed.

After the candidates have been pruned, they are sorted based on sample size, which is equal to the cluster size multiplied by the cluster number. Candidates are sorted in this manner to reduce the overall execution time. For example, assume two sampling regimen configurations pass the user-specified criteria for candidacy. Given that one may require 2000 clusters with a cluster size of 50,000 instructions, and another may require 40 clusters with a size of 1000 instructions, the latter should be chosen since it would require significantly less time to simulate.

The process of pruning sampling regimen candidates does not eliminate a cluster from being included in future sampling regimen configurations. The decision to prune a sampling regimen simply means that alternate cluster size and cluster number combinations should be explored.

5. Methodology

All experiments were conducted with the spec2000 benchmark suite. Integer benchmarks used included *gcc*, *mcf*, *parser*, *perl*, *vortex*, *vpr*, and *twolf*. Floating point benchmarks used included *ammp* and *art*. The first six billion instructions from each benchmark were simulated using reference input sets.

The model used in this study was an execution-driven simulator based on SimpleScalar [1]. The front end of the processor can fetch and dispatch eight instructions per cycle, and can issue and retire four instructions per cycle. The model included eight universal function units that were fully pipelined. The maximum number of in-flight instructions was 64. The issue queue size was 32, and there was a load-store queue of 64 elements. The pipeline depth was seven stages. The minimum branch miss-prediction penalty was five cycles. The processor frequency was assumed to be 2 GHz. The branch predictor was a 64K entry Gshare with an eight-entry return address stack. The BTB consisted of 4K entries. Architectural checkpoints were utilized to allow the processor to speculatively execute beyond eight branches.

A substantive memory hierarchy was modeled within the simulator. The first level data cache was 4-way and contained 32 KB with a 64-byte line size. The first level instruction cache was also 4-way and contained 64 KB with a 64-byte line size. The instruction and data caches were implemented using a *write-through-no-write-allocate* policy. The second level cache was 8-way and contained 1 MB with a 64-byte line size, and was implemented using a *write-back-write-allocate* policy.

A bus model also was incorporated in order to emulate arbitration, contention, and transfer delay between the levels of memory. The first level bus was shared between the first level data and instruction caches, and connected the first level caches to the second level cache. The first level bus had a width of 16 bytes and operated at 1 GHz. The second level bus connected the second level cache to main memory, had a width of 32 bytes, and operated at 2 GHz.

The model included both a functional and a timing simulator. The functional simulator was used to validate the results of the timing simulator. If the timing simulator attempted to commit a wrong value, the functional simulator would assert an error. However, in the context of sampled simulation, the functional simulator had additional uses. As instructions in the dynamic stream were skipped (either in cold or warm simulation), the functional simulator retained valid architectural state. When hot execution continued in the next cluster, the values of the registers contained in the functional simulator were copied to the timing simulator.

For processor simulations, the standard performance metric was IPC, which is the number of instructions retired per execution cycle.

6. Results

Figure 7 shows the time savings of single-pass simulation versus the total workload simulations as a factor speedup. As expected, 16.67%, or a sampling ratio of 1:6, had the least savings due to the size of the over-sample. All results in Figure 7 include both the profile sample simulation time and the second level simulation indicated by the regimen candidacy selection.

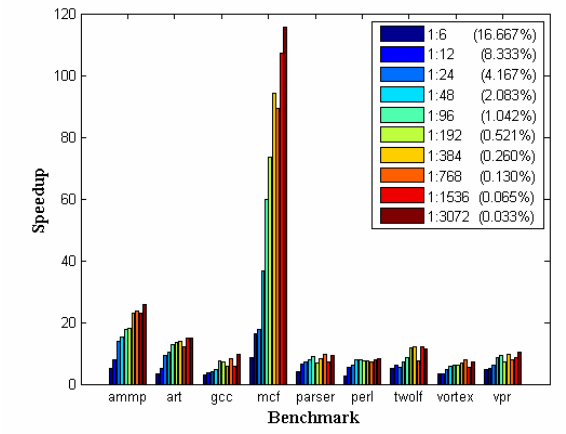

Figure 7

As previously stated, there were two types of biases that were introduced in the single-pass sampling methodology: 1) the introduced bias between the profile sample and the true execution; and 2) the introduced bias between the single-pass simulation and the profile sample. All second level simulations passed confidence tests when compared to the profile sample. At sampling ratios of 1:6, 1:12, 1:24, 1:48, 1:96, and 1:192, all second level simulations passed confidence tests when compared to the total execution.

Low sampling ratio simulations passed most of the confidence tests when compared to the total execution. At the sampling ratios of 1:384, 1:768, and 1:1536, 8 of 9 workloads passed confidence tests. At sampling ratios of 1:3072, 6 of 9 workloads passed confidence tests. This was expected behavior since as the profile sample decreased, so did estimate accuracy.

The lowest sampling ratio exhibited the highest factor speedup, with an average of 23.6 times faster, where *mcf* was 115 times faster than the entire workload simulation. When the sampling ratio of 1:192 was utilized a factor speedup of 16.85 times was obtained. In this study, a sampling ratio of 1:192 was optimal in achieving high speed without sacrificing accuracy (less than 1% error). Using this sampling ratio, the average number of clusters was 130, and the average cluster size was approximately 5000 instructions.

7. Related Work

The sampling of workloads has been used in a number of architectural simulation applications. Originally, sampling was applied to cache simulation [3],[7],[11],[21] and was later extended to the simulation of processors [5],[12],[17]. Most instances utilized some derivative of cluster sampling but other forms, such as stratified sampling [14] and set sampling [7],[10],[13], have been used with success.

Two different types of sampling are possible for caches: time sampling [4],[7],[11],[21] and set sampling [7],[10],[13]. Time sampling involves the extraction of time-contiguous memory references from different locations in an address trace. Set sampling is a form of stratified sampling when applied to caches and involves the inspection of particular cache sets. Thus, the memory references that affect chosen sets are not necessarily temporally adjacent.

Many different approaches have been used to remove non-sampling bias from sampled simulation. Laha, et al., [11] took sampling units immediately following context switches to ensure consistent state. By assuming the cache contents were flushed after a context switch for small caches, the contents were emptied, and were therefore identical to the full execution trace. For larger cache designs, the idea of primed cache sets was introduced by Fu, et al. [7] and Laha, et al. [11]. Once the execution of a new cluster began, a set in the cache was considered primed after it had been filled with unique references. Only information gathered from primed sets were used to record measurements. Other warm-up techniques proposed by Wood, et al. [21] use probability to distinguish misses at the beginning of a cluster between *compulsory* and *cold-start* misses.

Of all of the warm-up methods, perhaps the most accurate in removing non-sampling bias is SMARTS [20], proposed by Wunderlich, et al. When skipping instructions between clusters, the entire skip region of instructions is executed in a warm phase. Thus, every branch and memory operation is functionally applied to the branch predictor and cache hierarchy. The SMARTS warm-up policy has been applied in cache simulations [3] and to processor simulations [4],[5]. The SMARTS, or full functional, warm-up method is extremely accurate, but at a cost.

Because SMARTS is demanding in terms of simulation time, other warm-up methods have been proposed that approximate the SMARTS accuracy at

a lower cost. Haskins, et al. [8] proposed the *Memory Reference Reuse Latency (MRRL)* algorithm for warm-up. MRRL profiles the skip regions between clusters to determine the number of pre-cluster instructions to execute for a specified percentage warm-up. This work was later extended by Eeckhout, et al. [6] with the *Boundary Line Reuse Latency (BLRL)* algorithm. Unlike MRRL, BLRL only considers memory references from instructions that originate in the cluster. Another approach, known as *Reverse State Reconstruction (RSR)* [2], has been proposed to approximate SMARTS warm-up. In this technique, skipped instruction data is recorded and then applied to high-state structures in *reverse order*.

One widely popular approach used in lieu of statistical sampling was proposed by Sherwood, et al. [18]. This technique, called SimPoint, analyzes the frequency at which basic blocks are executed within a workload. From this heuristic, SimPoint identifies a region or set of regions that can be simulated to approximate the entire program behavior. This technique is hardware independent. While effective, critics of SimPoint note the heuristic by which the regions are selected utilizes systematic sampling. Since the probability of selection is not random, statistical tests such as the confidence interval cannot be used. An extension to SimPoint, called Variance SimPoint [16], has been introduced to calculate error bounds for sampled clusters. Such error bounds can only be calculated if SimPoint randomly selects clusters of execution.

Much work has been proposed with regards to warm-up methods, but there exists little work on sampling regimen design.

8. Summary

In this work, an efficient sampling regimen design process was presented. Utilizing a profile sample, thousands of sampling regimen configurations can be simultaneously evaluated for accuracy and statistical confidence. Each sampling regimen configuration was tested multiple times, each yielding a different random sample in order to increase the probability of correct classification. From the profile analysis, a list of possible sampling regimen configurations were identified and then pruned according to user-determined filter criteria. From this candidate list, sampling regimens were sorted based on sample size to allow the user to run the smallest, and therefore fastest, sample.

The techniques presented in this study are a vast improvement over traditional sampling regimen design. In this work, it was shown how sampling regimens can be determined being required to run the entire workload for accuracy comparisons. Significant time savings were realized since the entire workload was not required to be executed. Additionally, significant time-savings were realized by the efficient sampling regimen design algorithm.

9. References

[1] Burger, D. C., and Austin, T. M. *The Simplescalar Toolset, version 2.0*. Computer Architecture News, 25(3):13-25, 1997.

[2] Bryan, P. D., Rosier, M. C. Conte, T. M. *Reverse State Reconstruction for Sampled Microarchitectural Simulation*. IEEE ISPASS (San Jose, CA), April 2007.

[3] Conte, T. M., Hirsch, M. A., and Hwu, W. W. *Combining Trace Sampling With Single Pass Methods for Efficient Cache Simulation*. IEEE Transactions on Computers, Jun. 1998.

[4] Conte, T. M. *Systematic computer architecture prototyping*. PhD thesis, University of Illinois, Urbana, Illinois, 1992.

[5] Conte, T. M., Hirsch, M. A., and Menezes, K. N. *Reducing State Loss for Effective Trace Sampling of Superscalar Processors*. In Proc of the 1996 ICCD, (Austin, TX), Oct. 1996.

[6] EeckHout, L., Luo, Y., Bosschere, K. D., and John, L. K. *BLRL: Accurate and Efficient Warmup for Sampled Processor Simulation*. The Computer Journal, Vol. 48 (4). 2005.

[7] Fu, J. W. C., and Patel, J. H. *Trace driven simulation using sampled traces*. In Proc. 27th Hawaii Int'l. Conf. on System Sciences, (Maui, HI), Jan. 1994.

[8] Haskins, J. W., and Skadron, K. *Memory Reference Reuse Latency: Accelerated Sampled Microarchitecture Simulation*. In IEEE ISPASS, Mar. 2003.

[9] Henry, G. T. *Practical sampling*. Newbury Park, CA: Sage Publications, 1990.

[10] Kessler, R. E., Hill, M. D., and Wood, D. A. *A comparison of trace-sampling techniques for multi-megabyte caches*. IEEE Trans. Comput., vol. C-43, June 1994.

[11] Laha, S., Patel, J. A., and Iyer, R. K. *Accurate low-cost methods for performance evaluation of cache memory systems*. IEEE Trans. Comput., vol. C-37, Feb. 1988.

[12] Lauterbach, G. *Accelerating architectural simulation by parallel execution*. In Proc. 27th Hawaii Int'l. Conf. on System Sciences, (Maui, HI), Jan. 1994.

[13] Lui, L., and Peir, J. *Cache sampling by sets*. IEEE Trans. VLSI Systems, vol. 1, June 1993.

[14] Mangione-Smith, W. H., Abraham, S. G., and Davidson, E. S. *Architectural vs Delivered Performance of the IBM RS/6000 and the Astronautics ZS-1*. In Proc. 24th Hawaii International Conference on System Sciences, January 1991.

[15] McCall, J. C. H. *Sampling and statistics handbook for research*. Ames, Iowa: Iowa State University Press, 1982.

[16] Perelman, E., Hamerly G., and Calder, B. *Picking Statistically Valid and Early Simulation Points*. In the International Conference on Parallel Architectures and Compilation Techniques, 2003.

[17] Poursepanj. *The PowerPC performance modeling methodology*. Communications ACM, vol. 37, pp. 47–55, June 1994.

[18] Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. *Automatically Characterizing Large Scale Program Behavior*. In the 10th ASPLOS, October 2002.

[19] Wenisch, T. F, Wunderlich, R. E., Falsafi, B., and Hoe, J. C. *Simulation Sampling with Live-Points*. IEEE ISPASS, Mar. 2006.

[20] Wunderlich, R. E., Wenish, T. F, Falsafi, B., and Hoe, J. C. *SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling*. Proc. 30th ISCA, 2003.

[21] Wood, D. A., Hill, M. D., and Kessler, R. E. *A model for estimating trace-sample miss ratios*. In Proc. ACM SIGMETRICS '91 Conf. on Measurement and Modeling of Comput. Sys., May 1991.