Reducing State Loss For Effective Trace Sampling of Superscalar Processors

Thomas M. Conte Mary Ann Hirsch Kishore N. Menezes
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina 27695
{conte, mahirsch, knmeneze}@eos.ncsu.edu

Abstract

There is a wealth of technological alternatives that can be incorporated into a processor design. These include reservation station designs, functional unit duplication, and processor branch handling strategies. The performance of a given design is measured through the execution of application programs and other workloads. Presently, trace-driven simulation is the most popular method of processor performance analysis in the development stage of system design. Current techniques of trace-driven simulation, however, are extremely slow and expensive. In this paper, a fast and accurate method for statistical trace sampling of superscalar processors is proposed.

1 Introduction

There is a wealth of technological alternatives that can be incorporated into a processor design. These include reservation station design, functional unit duplication, processor branch handling strategies, and instruction fetch, issue, completion, and retirement policies. The performance of a given design is measured through the execution of application programs and other workloads. The decision to introduce a new technology improvement in a system depends, however, on the relative performance increase versus the cost.

Workloads or benchmarks may be instrumented to generate traces that contain enough instruction execution information to test the performance of the proposed processor subsystem. The SPEC92 suite [5] is a set of benchmarks that has been widely used to measure such performance. These benchmarks execute for billions of instructions; therefore, an exhaustive search of the design space using these workloads is time-consuming and expensive. Statistical sampling [8],[12], has been used successfully to alleviate these problems in cache simulations and, in recent years, it has also been extended to the simulation of processors [2],[10],[9]. In general, statistical techniques used in experiments reduce a large data set into a smaller representative set. The results obtained from the sampling must be representative of the entire workload to be accurate. The representativeness, therefore, depends on the method of sample collection. One of the earliest studies of trace-sampled processor simulation was done by Conte [2], which showed that a systematic sampling method that utilized contiguous traces could estimate processor performance with a relative error of only 13% [2]. Poursepanj later successfully developed a performance modeling technique for the PowerPC 603 microprocessor that employed the use of a sample of one million instructions equally divided into 200 clusters [10]. At the same time, Lauterbach proposed an iterative sampling-verification-resampling technique for processor performance simulation [9]. This sampling method extracted 100 clusters at random intervals with each cluster consisting of $100,000$ instructions. These selected clusters are checked against the full trace to monitor the sample's representativeness of the instruction frequencies, basic-block densities, and cache statistics. If the representativeness of the sample does not fall within the specified criterion, more clusters are selected. This selection process continues until the required criterion is reached [9]. All of these previous techniques were relatively accurate but these methods required full traces and were therefore extremely long and costly.

In this paper, a fast and accurate method for statistical trace-sampling for processor simulation, the *state-reduction* method, is proposed. This method can be used to design a sampling regimen without the need for full-trace simulations. In order to achieve this goal, statistical metrics are used in this technique to derive the sampling regimen and predict the accuracy of the estimated results. Section 2 provides a discussion of statistical sampling techniques and sample design for processor simulation. The following section describes the *state-reduction* method in detail and includes a statistical evaluation of its performance. The paper concludes with a summary of the *state-reduction* technique and a discussion of future work.

2 Statistical Sampling

Sampling has been defined as the process of reducing a large data set into a smaller representative set [1]. Sampling may be broadly classified into two types: *probability sampling* and *non-probability sampling*. Probability sampling requires that samples are chosen by a randomized mechanism which assures the selection of samples that is independent of subjective judgments. *Simple random sampling* is known to be one of the most accurate methods for sampling large populations. This technique involves a random selection of single, independent elements from the population. However, the cost associated with this approach makes its application infeasible in some cases. Another less accurate, but cost-effective technique is *cluster sampling*. This method collects contiguous groups of elements at random intervals from the population. Cluster sampling is the sampling technique commonly applied to processor simulation [2].

A *sampling unit* is an element which has an interesting property that should be studied. The sampling unit for a processor is a single execution cycle of the processor pipeline. A *sample* is defined as the total number of sampling units from which the performance metric is calculated. The accuracy of the sampled results, therefore, increases as the size of the sample increases. Larger samples, however, also increase the time and cost of simulation, thereby making the choice of an efficient sample size critical.

A *parameter* is an interesting population characteristic. The primary parameter for processor simulation is the *mean retired instructions per cycle* (IPC). Consider a processor running a benchmark which executes in n time cycles, $t, t+1, t+2, \ldots, t+n$, where $t+i$ is a single execution cycle. For a processor, these execution cycles constitute a complete list of the sampling units known as the *total population*. In traditional processor sampling, sampling units of equal length are randomly selected as the entire trace is traversed. This method does not, however, yield any savings in simulation cost because a simulation of the full trace is required. An alternative approach, *cluster sampling*, extracts subsets of the trace at random gaps during trace generation.

The sampled unit of information for processor simulation is the number of instructions executed during a processor simulation. The metric is measured for each execution cycle and is called the *instructions/cycle* (IPC). The IPC, however, may vary between simulation runs. This variation requires the results to be validated. The validation of estimated IPC is calculated as the *relative error*, RE(IPC) which is defined as:

$$\text{RE(IPC)} \quad = \quad \frac{\mu_{IPC}^{true} - \mu_{IPC}^{sample}}{\mu_{IPC}^{true}} \qquad (1)$$

where μ_{IPC}^{true} is the true population mean IPC, and μ_{IPC}^{sample} is the sample mean IPC.

2.1 Sample design

Sample design involves the choice of a robust (*i*) *sample size*, (*ii*) *cluster size* and, (*iii*) *number of clusters*. The accuracy of estimates for a particular sample design is primarily affected by two kinds of bias [7]: *sampling bias* and *nonsampling bias*.

Nonsampling bias arises when the population being sampled (*the study population*) is different from the *actual target* population. Nonsampling bias in processor simulation is primarily due to the *cold-start effect*. The state of a processor is composed of many units including the reservation stations, functional unit pipelines, and branch handling hardware. The cold start effect occurs when sampling is employed and clusters are extracted from different locations in the full trace. This situation then causes the states of the units to be different from the real states produced by the full trace simulation.

Sampling bias is measured as the difference between the mean of the sampling distribution and the sample mean. Clusters from different locations may be selected from sample to sample which causes the estimates to vary across repeated simulations. Therefore, the estimates are dependent on the sample design. The means of these samples form a distribution that is known as the *sampling distribution*. Statistical theory states that, for a well designed sample, the mean of the sampling distribution is representative of the true mean, and, the accuracy of the estimated mean increases as the sample size increases. In case of cluster sampling, this *sample size* is defined as the product of the *number of clusters* times the *cluster size*, yet the only randomness in cluster sampling is due to the number of clusters. The reduction of sampling bias in cluster sampling requires that the design of the sample be robust and include all factors that could increase error. Some of the methods that have been used to overcome or reduce the total bias are discussed in the following sections.

2.2 The Processor Model

The processor model must be considered when a sampling regimen is designed. In this study, a highly-parallel processor model is used to develop a robust nonsampling bias reduction technique. In addition, the model is used to test the sample design strategy. The processor model considered is a full-Tomasulo, *out-of-order* execution engine that is based on a RISC design methodology. This model assumes a perfect cache and has 7 different types of functional units (see Table 1). Processor instructions are issued at an aggressive rate of eight instructions/cycle. In addition, this model contains multiple copies of key functional units, and each of these functional units has access to an unlimited supply of reservation stations. Highly-accurate branch prediction and speculative execution are generally accepted as essential for high superscalar performance, so a hardware predictor with high prediction accuracy is incorporated in this processor model. Specifically, the *two-level adaptive training branch predictor* [13] is employed. This scheme consists of a 1024-entry table known as the *History Register Table* (HRT), which maintains a history of the last eight executions of a branch. The entries in this HRT point to locations in another 1024-entry table called the *Pattern Table* (PT). A branch prediction is made using a 2-bit counter predictor in the PT. In addition, this processor is able to use the results of the predictor to speculatively execute beyond three branches. In this paper, the entire branch prediction hardware including the HRT/PT tables for this model will be referred to as the *branch history buffer* (BHB).

Table 1. Processor model design parameters.

Issue rate: 8 *instructions/cycle*			
Scheduling: *Full-Tomasulo, out-of-order*			
Branch handling: *Two-level adaptive training* ("PAs")			
Branch speculation degree: 3 *branches ahead*			
Functional unit	Description	Number	Latency
Alu	Arithmetic logic unit	4	1
Load	Load	8	2
Store	Store	64	1
FPAdd	FP add	3	2
FPMul	FP multiply	3	3
FPDiv	FP divide, remainder	1	13
Branch	Branch	3	1

* FPDiv is unpipelined.

Table 2. Relative error for a 10 million instruction single-cluster sampling technique.

Benchmark	μ_{IPC}^{true}	μ_{IPC}^{sample}	RE(IPC)
compress	2.786	3.207	-15.11
eqntott	2.523	3.072	-21.76
espresso	2.440	2.879	-17.99
gcc	2.574	2.336	9.25
li	2.481	2.510	-1.17
sc	2.214	3.358	-51.67
doduc	3.425	3.465	-1.17
mdljsp2	2.545	1.902	25.27
ora	2.932	2.932	0.0
tomcatv	4.964	5.949	-19.84
		average:	16.32%

3 Trace Sampling for Processors

Many published studies of instruction-level parallelism calculate the IPC based on a single cluster selected from the beginning of a benchmark's execution. This approach has many drawbacks. Table 2 presents the results of simulations using the first 10 million instructions from the trace of each of the benchmarks studied. In this research, the benchmarks include all of the SPECint92 benchmarks (*compress, espresso, eqntott, gcc, li*, and *sc*) and four of the SPECfp92 benchmarks (*doduc, mdljsp2, ora*, and *tomcatv*). Several of these benchmarks achieve relatively accurate results with this single cluster technique. Yet, the μ_{IPC}^{sample} of *sc* is overestimated in excess of 50%, and the majority of other benchmarks experience errors in excess of 15%. These overestimations are due to the fact that a single cluster is rarely representative of an entire benchmark. This single cluster is selected at the very beginning of the trace, therefore it may contain all of the instructions for setting up later code execution sequences. This additional code rarely contains instructions that are executed repeatedly (i.e. loops) or have dependencies. The impact of this lack of dependencies causes the estimated IPC to be higher than the real IPC.

While previous studies have tried to reduce all bias as a whole and make a prescription for *all* trace-sampled processor simulation, this study separates bias into its nonsampling and sampling components. This approach solves the statistical sampling problems by analyzing the effects of both sampling and nonsampling bias. In the first step of the sampling process, clusters of instructions are obtained at random intervals and written to a disk file. The term *cluster* is used interchangeably for the group of instructions that yields a set of contiguous execution cycles, and for the set of execution cycles themselves. The number of execution cycles that would be obtained on the execution of a sampled instruction trace, N_E^{sample}, is given by,

$$N_E^{sample} = \frac{N_I^{cluster} \times N_{cluster}}{\mu_{IPC}} \qquad (2)$$

where $N_I^{cluster}$ is the number of instructions in a cluster, $N_{cluster}$ is the number of clusters, and μ_{IPC} is the mean IPC. In the following sections, a detailed description of the techniques employed for reducing sampling and nonsampling bias are given. These techniques are known as the *state-reduction* method.

3.1 Reduction of nonsampling bias

In the case of processor simulation, nonsampling bias is due to any loss of state information. The state in a processor is kept in the reservation stations, the functional unit pipelines, and in the branch handling hardware (BHB). For the processor model considered in this research, the largest amount of state is contained in the BHB. Therefore, the *state-reduction* strategy will focus on the accuracy of BHB state.

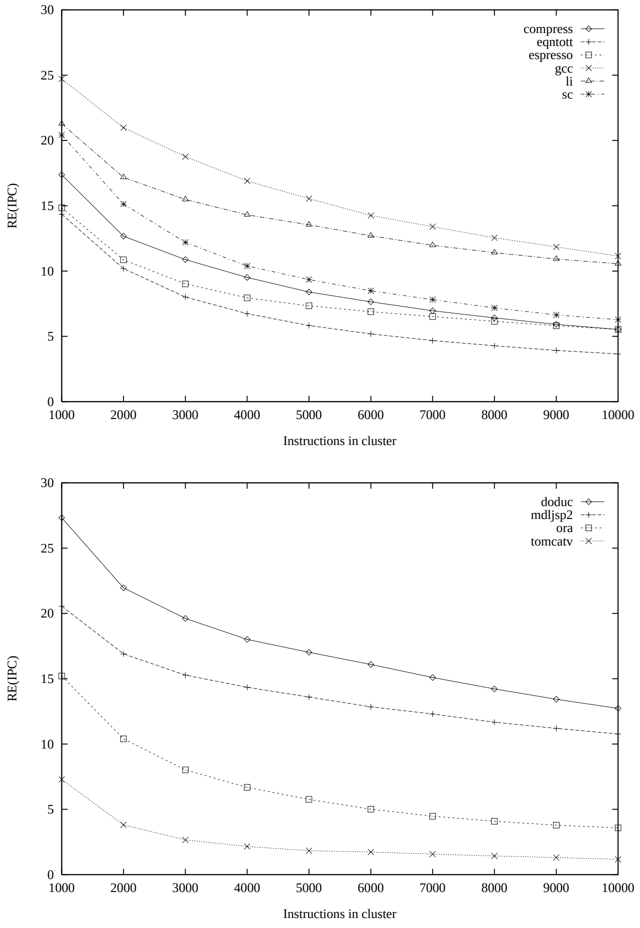

Figure 1. Relative error vs. Cluster size (*fresh-BHB*, $N_{cluster}$ = $2,000$)

A study of the nonsampling bias for the processor model is shown in Figure 1. For these experiments, the number of clusters considered is $2,000$, which is large enough that it does not to contribute the error. The size of each cluster is then varied from 1,000 to 10,000 instructions. Two different state repair technique for processor sampling were studied. In *fresh-BHB*, the BHB was completely flushed between the simulation of each cluster. The experimental results for *fresh-BHB* are shown in Figure 1. The relative error is high when *fresh-BHB* is used. For example, it can be as high as 27.33% for *doduc*, using a cluster size of 1,000 instructions. The cluster size and the RE(IPC) are inversely related because the BHB warms up the empty buffer with the initial part of each cluster. It is known that the execution of branch instruction is not independent and the execution time of a branch can vary. The *fresh-BHB* technique, however, incorrectly assumes that a branch will only execute during a given cluster (i.e. it will not execute during the beginning of the next cluster). The *stale-BHB* method takes the opposite approach. The *stale-BHB* technique assumes that all of the branches that were active in the previous cluster are active in the beginning of the correct cluster. The results for the *stale-BHB* approach are presented in the Figure 2. The behavior pattern for *stale-BHB* is similar to the *fresh-BHB*. The *stale-BHB*, however, has considerably lower overall RE(IPC) for all of the benchmarks studied.

Table 3. Dynamic branch distributions.

The *E-x* columns present the number of branches that occupy x percent of the benchmark's execution. E-100 is the total number of branches that are dynamically executed. *Branches in program* is the total number of static branches in the program text.

	Branch instructions					
Benchmark	E-25	E-50	E-90	E-99	E-100	Branches in program
compress	2	5	17	21	135	432
eqntott	1	2	6	34	502	1323
espresso	15	49	225	842	2838	7582
gcc	72	348	2610	6535	14382	34347
li	10	34	119	264	1058	3138
sc	2	7	52	135	1529	4634
doduc	1	7	283	468	1596	3643
mdljsp2	2	5	15	35	821	848
ora	3	6	13	24	396	1791
tomcatv	3	6	12	14	372	1318

The characteristics of the branch instructions in the benchmarks presented in Table 3 explain much of the behavior seen in Figure 1. The data in Table 3 represents the distribution of unique branch instructions during execution. The "E-x" column presents the number of branches that occupy x percent of the benchmark's execution. For example, of the 1323 branches in eqntott, only 502 are actually executed. Of these, only one branch accounts for 25% of the execution time and two branches account for 50% of the execution. These results show that the benchmarks exercise only a very small number of dynamic branches for the majority of their execution. Logically, the benchmarks in this study can be divided into three categories based on the number of branches executed and generally accepted statistical principles. The categories are *easily sampled*, *intermediate*, and *difficult to sample*. The benchmarks, *compress*, *eqntott*, *ora*, and *tomcatv*, fall into the easily sampled category because only a relatively small number of branches are executed. This implies that the probability for obtaining a representative sampling is high. The second category, intermediate, contains *espresso*, *li*, *sc*, *doduc*, and *mdljsp2*.

All of these benchmarks have a large number of branches executed therefore a representative sampling is harder to obtain. The difficult to sample category only contains *gcc*. This benchmark has a very large number of branches executed therefore the chance of obtaining a representative sampling with a small number of clusters is very low. The RE(IPC) for *fresh-BHB* is shown in Figure 1. These results (see Figure 1), however, do not coincide with the sampling difficulty categories. The difficulty of sampling, in general, is not directly correlated to the RE(IPC). The RE(IPC) ranges from 7% to 17% for easily sampled benchmarks, yet the RE(IPC) for immediate benchmarks ranges from 15% to 27%. In addition, the most difficult to sample benchmark *gcc* has an RE(IPC) of approximately 25%. The RE(IPC) results for *stale-BHB* presented in Figure 2 show a similar pattern. The RE(IPC) of easily sampled benchmarks range from 2% to 7% while the RE(IPC) for intermediate benchmarks ranges from 5% to 8%. Although in the case of the benchmark, *gcc*, the RE(IPC) is 15% which is well beyond the range of the intermediate benchmarks' RE(IPC). The *gcc* phenomenon may be accounted for by the choice of state repair techniques.

Another classification scheme for the benchmarks studied is based on the lifetime of branch execution. This approach categorizes benchmarks based on the relative lifespans of their component branches. This methodology implies that the choice of state repair technique will have more impact on the accuracy of performance estimates when a benchmark has more branches that have longer overall lifetimes (i.e. the branch's execution occurs during more than one cluster). When the results from the *fresh-BHB* and *stale-BHB* techniques are compared, the importance of branch execution lifetime and state repair technique can be seen. If the *fresh-BHB* method, all states are cleared at the beginning of each cluster. This approach totally ignores the lifetime of the branch. In fact, the *fresh-BHB* method assumes that a branch's lifetime will be shorter than the combined length of a cluster and a gap. Table 3, however, indicates that the lifetime of certain branches may be longer. The *stale-BHB* approach, applies the alternative philosophy. In this technique, it is assumed that a branch that was active in the previous cluster will be active in the current cluster (i. e. no state loss). The experimental results indicate that the *stale-BHB* approach is more accurate than the *fresh-BHB* method. These results imply that very few branches in a benchmark need to have long lifetimes in order to substantially impact RE(IPC) if the incorrect state repair technique is selected. For example, the benchmark, *doduc*, has only 0.1% of its branches having a long execution lifetime. The use of *stale-BHB* on *doduc*, however, can reduce the RE(IPC) from approximate 27% to less than 8%.

In the *stale-BHB*, the state of the BHB at the end of a cluster was used as the state of the BHB at the beginning of

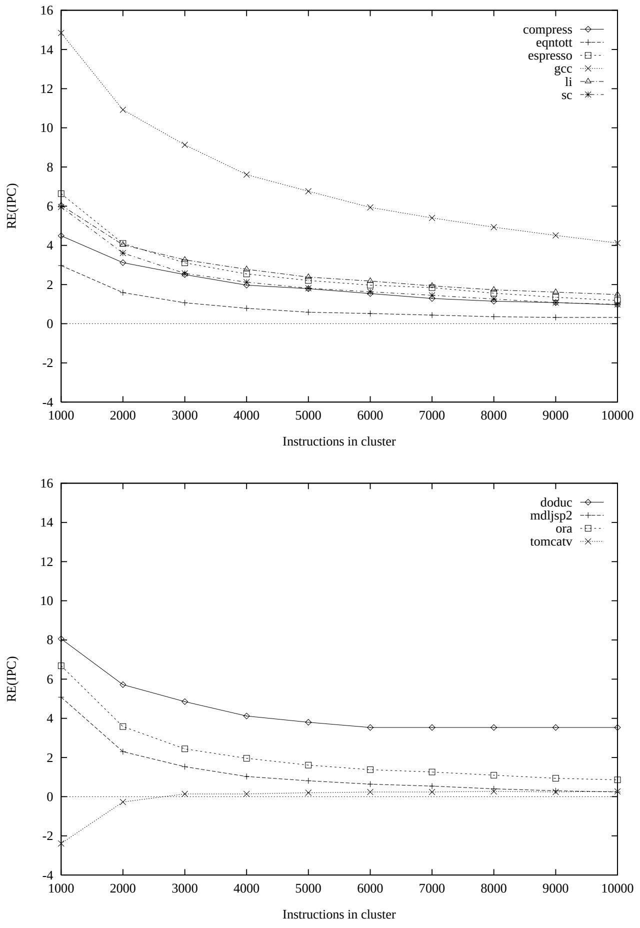

Figure 2. Relative error vs. Cluster size (*stale-BHB*, $N_{cluster}$ = $2,000$)

the next cluster. This method assumed that no substantial changes in BHB state occurred during the gap, which is known not to be the case. In an effort to compensate for the effects of state change during the gap, the use of a warm-up period has been suggested [8]. During a warm-up period, the instructions contained in a cluster are applied to the BHB but no statistics (i.e. IPC) are calculated. In essence, the warm-up period tries to stabilize the state of the BHB before any calculations are performed. In this research, the effects of a warm-up period in conjunction with the *stale-BHB* technique are studied. The effects of warm-up on RE(IPC) were studied by prepending variable numbers of warm-up instructions to each cluster. The warm-up period during each sample contains a fixed number of instructions. The results from this approach are given in Figure 3 and 4. These results indicate that RE(IPC) decreases as the amount of warm-up increases, regardless of the cluster size. In general, the RE(IPC) stabilizes at the $7,000$ instructions level. This phenomenon supports the conjecture that the closer the simulated state of the BHB is to the true state (calculated by a full trace), the more accurate the estimated IPC will be.

In summary, reduction in nonsampling bias can be achieved by manipulating three parameters: state repair method, cluster size and warm-up period. According to the experimental results, the most accurate IPC estimation is achieved using *stale-BHB*, a cluster size of at least $2,000$ instructions, and a warm-up period which contains a minimum of $7,000$ instructions. In addition, these small cluster and warm-up period sizes indicate that the *state-reduction* approach produces a significant reduction in simulation execution. If this method is employed, the IPC is accurately estimated in a fraction of the execution time required by a full simulation. These results, however, only apply to a highly parallel processor model that includes a large branch predictor. Current processors designs contain a smaller amount of state information. Therefore, this sample design strategy will be more robust if a current processor design is simulated. (A discussion of sample design for an arbitrary processor model is presented in section 4.)

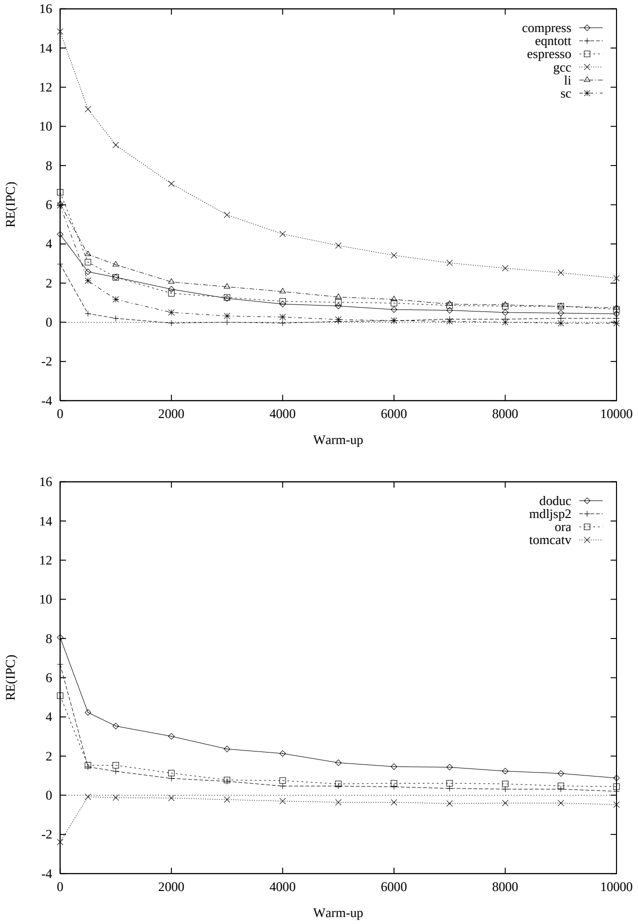

Figure 3. Relative error vs. Warm-up (*fixed*, $N_{cluster}$ = $2,000$, cluster size = $1,000$, *stale-BHB*).

3.2 Reduction in sampling bias and variability

In sampling theory, it is known that bias exists in every sample because of the random nature of the sample. The extent of the error caused by this bias, however, can be predicted. The *standard error* of the statistic under consideration is used to measure the precision of the sample results [1]. Standard error is a measure of the expected variation between repeated sampled simulations using a particular sampling methodology. It is not cost-effective to perform repeated sampled simulations to measure the standard error. Sampling theory, however, allows the estimation of the stan-

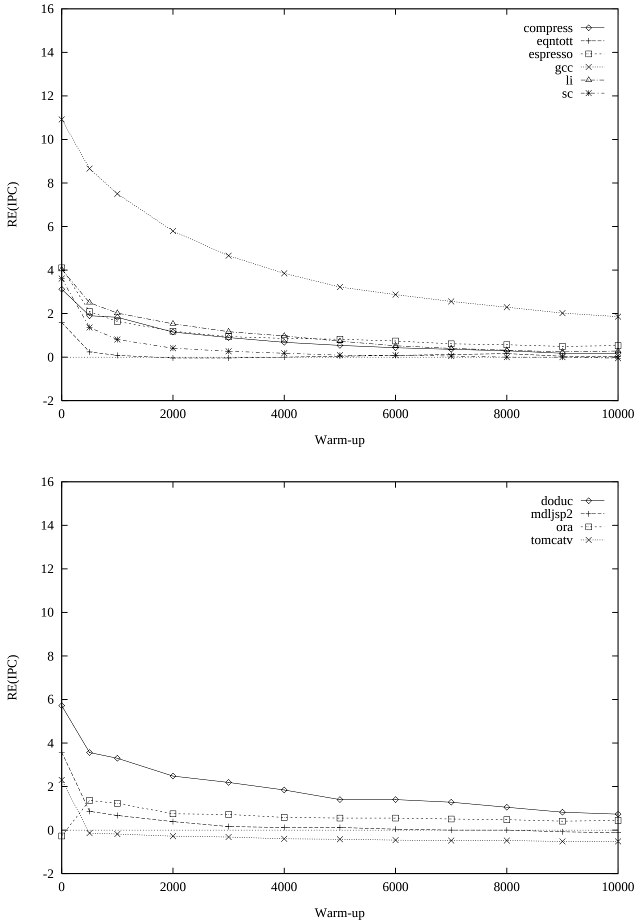

Figure 4. Relative error vs. Warm-up (*fixed*, $N_{cluster}$ = $2,000$, cluster size = 2,000, *stale-BHB*).

dard error ($S_{\overline{IPC}}$) from a single simulation. The standard deviation for a cluster sampling design is given by,

$$s_{IPC} = \sqrt{\frac{\sum_{i=1}^{N_{cluster}} (\mu_{IPC}^{i} - \mu_{IPC}^{sample})^2}{(N_{cluster} - 1)}}, \qquad (3)$$

where μ_{IPC}^{i} is the mean IPC for the i_{th} cluster in the sample. The estimated standard error can then be calculated from the standard deviation for the sample as,

$$S_{\overline{IPC}} = \frac{s_{IPC}}{\sqrt{N_{cluster}}}. \qquad (4)$$

The estimated standard error can be 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* $S_{\overline{IPC}}$, where the error bound is $1.96S_{\overline{IPC}}$. Moreover, for a well designed sample, with negligible nonsampling bias, the true mean of the population may also be expected to fall within this range. Low standard errors imply little variation in repeated estimates and consequently result in higher accuracy.

The relationship between sample size and standard error is presented in Figure 5. These results indicate that the standard error generally decreases as the number of clusters increases. However, the data also shows that there is little if any decrease in standard error if the number of clusters is greater than 700. The floating-point benchmarks have larger standard errors at a smaller number of clusters than integer benchmarks do. By this fact, the rest of this study will set the number of clusters to be $1,000$ for further analysis.

Table 4. Confidence interval measurements from estimates obtained from single samples ($N_{cluster}$ = 1,000).

Benchmark	True mean (μ_{IPC}^{true})	Estimated mean (μ_{IPC}^{sample})	Standard Error ($S_{\overline{IPC}}$)	Relative Error (RE(IPC))
compress	2.786	2.768	0.016	0.65
eqntott	2.523	2.521	0.007	0.08
espresso	2.440	2.414	0.017	1.07
gcc	2.574	2.498	0.039	2.95
li	2.481	2.488	0.012	-0.28
sc	2.214	2.220	0.009	-0.27
doduc	3.425	3.391	0.050	0.99
mdljsp2	2.545	2.551	0.035	-0.24
ora	2.932	2.919	0.003	0.44
tomcatv	4.964	4.983	0.039	-0.38

The values of $S_{\overline{IPC}}$ for a sample made up of 1,000 clusters are presented in Table 4. *Doduc* has the maximum standard error and the largest error bounds. Its confidence interval indicates that the mean IPC for repeated samples should be between 3.293–3.523 ($\mu_{IPC}^{sample} \pm 1.96S_{\overline{IPC}}$) to

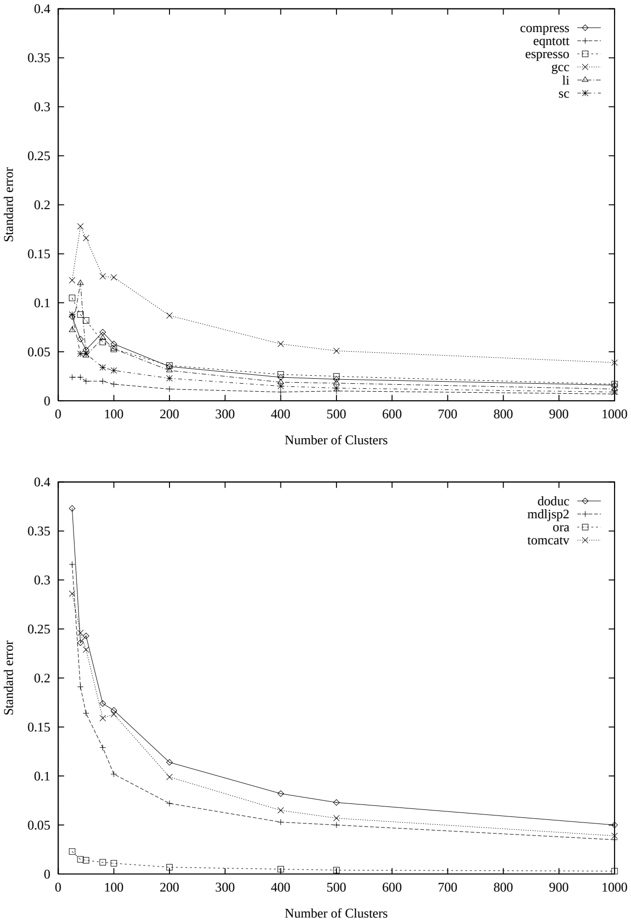

Figure 5. Standard Error vs. Number of Clusters (*Stale-BHB*, $N_{cluster}$ = 2000, cluster size=2000, warm-up=8000)

obtain a 95% confidence interval. These results also show that the μ_{IPC} for all of the benchmarks with the exception of *ora* fall within the 95% confidence level. The μ_{IPC}^{true} for *ora* is approximately 0.01% out of its 95% confidence level. This deviation could be due to the lack of a wide variation in its sample IPC means, therefore its standard error is relatively small error. If these samplings where repeatedly performed, the μ_{IPC}^{sample} for *ora* may fall within the overall 95% confidence level.

The variability of cluster means across all clusters in a sample is presented in the Figure 6. The results show that the cluster means vary with the number of clusters used. The difficulty of accurately sampling a given benchmark is inversely related to the variability of the cluster means. Benchmarks, *ora* and *eqntott*, exhibit little variability and are therefore conducive to accurate sampling. The benchmarks, *gcc*, *doduc*, *mdljsp2*, and *tomcatv*, exhibit high variation in the cluster means and are therefore difficult to sample. These results indicate that the precision of a sampling regimen depends upon the homogeneity of the cluster means.

In this study, the full-trace simulations were also available, thereby making it possible to test whether sample design using standard error achieves accurate results. The estimates of μ_{IPC}^{sample} when compared to μ_{IPC}^{true} show relative errors of less than 3% for all benchmarks (see Table 4). Furthermore, the relative error was predicted by the 95% confidence levels. Therefore, a robust sampling regimen can be designed without the need for full-trace simulations. In addition, if nonsampling bias is negligible, the sampling regimen can be designed from the data obtained solely from a single sampled run.

4 Concluding Remarks

Trace-driven simulation is a popular approach to evaluating processor design alternatives. This evaluation, however, traditionally involves the use of costly full trace simulations. The need for efficient processor sampling techniques has become more apparent due to the fact that the execution time for a full continuous trace is extremely long.

In this paper, a fast and accurate processor sampling design strategy, the *state-reduction* method, has been presented that does not require full trace simulation. The essential steps to this strategy are centered around the systematic reduction in sampling and nonsampling bias. To reduce sampling bias, statistical sampling design techniques were employed. The experimental results demonstrate that a regimen for sampling a processor simulation can be developed without the need for full-trace simulations. The nonsampling bias is reduced by using a combination parameters that include a state repair method, the cluster size, and a warm-up period. The recommended steps for processor sampling design using the *state-reduction* technique are:

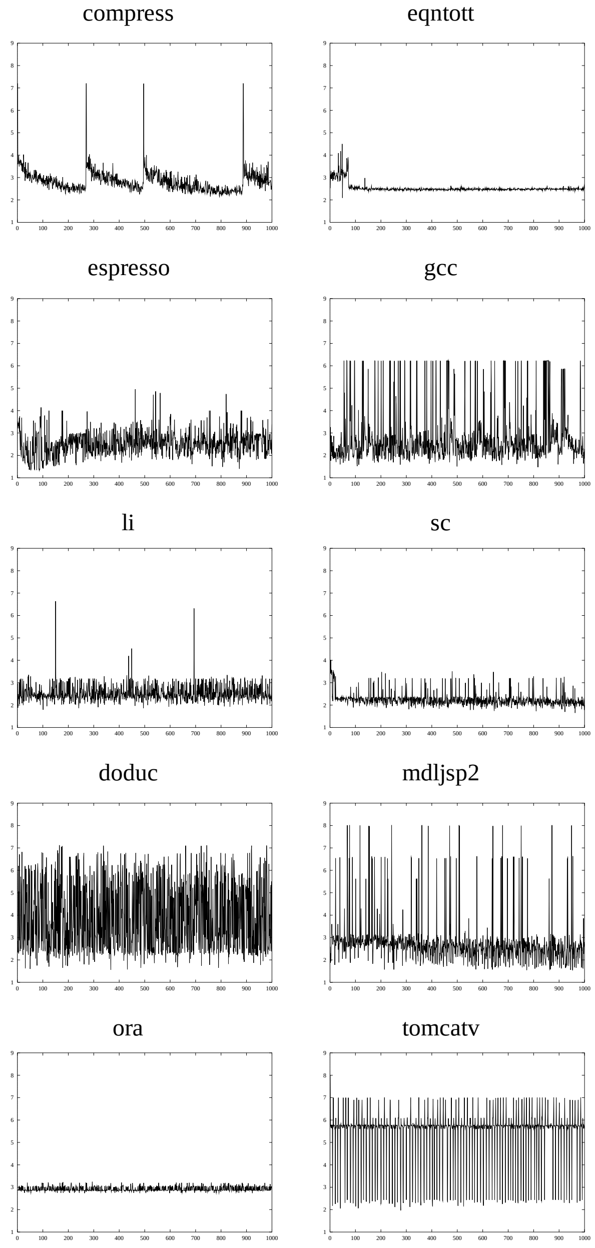

Figure 6. Variability of cluster means across all clusters in the samples. X-axis is the cluster number and y-axis is the mean IPC for the cluster. Cluster size = 1000.

1. **Reduce nonsampling bias:** This requires the selection of a state repair mechanism, a cluster size, and a warm-up period.

2. **Determine the sample design:**

   (a) **Select a number of clusters:** Simulate using a particular number of clusters.

   (b) **Determine error bounds:** Estimate standard error (Equations 3 and 4) to determine error bounds/precision of the results. If the error is acceptable, the experiments are completed. Otherwise, increase the sample size by increasing the number of clusters, and resimulate until the desired precision is achieved.

The *state-reduction* method not only can be used for sampling in processor simulation. It can also be extended to processor emulation as developed by Sathaye [11]. Clearly, the *state-reduction* technique can easily be supported by generic processor simulation generators such as the VMV proposed by Diep [3], [4].

Future work is needed that will incorporate the use of static branch predictors into the warm-up period for each cluster. This implementation would extend the Fu and Patel prediction technique [6] for cache simulation to processor simulation. In the current *state-reduction* method, the IPC is not calculated during the initial warm-up period. The future extension of this method, however, would calculate the IPC during the warm-up period based on a branch instruction path prediction. During the evaluation interval, the IPC would still be calculated on the actual cluster instruction information. This modified *state-reduction* method may increase the accuracy of the estimated IPC without severely degrading the speedup achieved by the original technique.

References

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

[2] T. M. Conte. *Systematic computer architecture prototyping*. PhD thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, Illinois, 1992.

[3] T. A. Diep. *VMW: A Visualization-based Microarchitecture Workbench*. PhD thesis, Carnegie Mellon University, Pittsburgh, Pennsylvania, June 1995.

[4] T. A. Diep, C. Nelson, and J. P. Shen. Performance evaluation of the powerpc 620 microarchitecture. In *Proc. 22th Ann. Int'l. Symp. Computer Architecture*, pages 163–174, June 1995.

[5] K. M. Dixit. CINT92 and CFP92 benchmark descriptions. *SPEC Newsletter*, 3(4), 1991. SPEC, Fairfax, VA.

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

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

[8] S. Laha, J. A. Patel, and R. K. Iyer. Accurate low-cost methods for performance evaluation of cache memory systems. *IEEE Trans. Comput.*, C-37(1):1325–1336, Feb. 1988.

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

[10] A. Poursepanj. The PowerPC performance modeling methodology. *Commun. ACM*, 37(6):47–55, June 1994.

[11] S. W. Sathaye. Mime: A tool for random emulation and feedback trace collection. Master's thesis, Department of Electrical and Computer Engineering, University of South Carolina, Columbia, South Carolina, 1994.

[12] H. S. Stone. *High-performance computer architecture*. Addison-Wesley, New York, NY, 1990.

[13] T. Yeh. *Two-level adaptive branch prediction and instruction fetch mechanisms for high performance superscalar processors*. PhD thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, 1993.