TSS: Applying Two-Stage Sampling in Micro-architecture Simulations

Zhibin Yu, Hai Jin
Service Computing Technology and System Lab
Cluster and Grid Computing Lab
Huazhong University of Science and Technology
Wuhan, 430074, China
{yuzhibin, hjin}@hust.edu.cn

Jian Chen, Lizy K. John
Department of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712 USA
{jchen2, ljohn}@ece.utexas.edu

Abstract—**Accelerating micro-architecture simulation is becoming increasingly urgent as the complexity of workload and simulated processor increases. This paper presents a novel two-stage sampling (TSS) scheme to accelerate the sampling-based simulation. It firstly selects some large samples from a dynamic instruction stream as candidates of detail simulation and then samples some small groups from each selected first stage sample to do detail simulation. Since the distribution of standard deviation of cycle per instruction (CPI) is insensitive to micro-architecture, TSS could be used to speedup design space exploration by splitting the sampling process into two stages, which is able to remove redundant instruction samples from detail simulation when the program is in stable program phase (standard deviation of CPI is near zero). It also adopts systematic sampling to accelerate the functional warm-up in sampling simulation. Experimental results show that, by combining these two techniques, TSS achieves an average and maximum speedup of 1.3 and 2.29 over SMARTS, with the average CPI relative error is less than 3%. TSS could significantly accelerate the time consuming iterative early design evaluation process.**

Keywords:Micro-architecture Simulation; Two-stage Sampling; Functional Warm-up; Performance Evaluation

I. Introduction

Computer architects heavily rely on cycle-accurate micro-architecture simulators to evaluate the performance and power of different configurations during early design stages. The iterative nature of the evaluation process underscores the importance of detail simulation speed, which is still prohibitively low and fundamentally constraints the scope and the efficiency of early design evaluation. It may take a state-of-the-art simulator days or even weeks to simulate real-world workloads. This problem is exacerbated as the size of representative workloads continuously increasing (e.g., the dynamic instruction count of SPEC CPU2006 is ten times that of SPEC CPU2000). Therefore, it becomes increasingly urgent to accelerate micro-architecture simulations without losing its accuracy and representativeness.

Previous research on time-efficient simulation can be categorized into three dimensions, i.e., input dataset reduction, benchmark subsetting, and program phase sampling. Input dataset reduction schemes, such as MinneSPEC [1], attempt to reduce the dynamic instruction count of a benchmark program by substituting the original large dataset with a smaller dataset. The benchmark subsetting technique tries to remove the inter-program redundancy in a benchmark suite by using only a few representative programs, as opposed to entire benchmark suite, to do architecture evaluation [2][3]. The program phase sampling method, however, explores intra-program redundancy and accelerates simulation by choosing only portions of the program's dynamic instruction stream for detail simulation. Depending on the sampling style, this method could be further classified as representative sampling, such as SimPoints [4][5], and systematic sampling, such as SMARTS [6]. Among these above methods, the program phase sampling approach is probably most widely used since it provides high simulation accuracy and speedup without undermining the representativeness of the workloads. Therefore, this paper focuses on improving the sampling-based simulation method, in particular, the SMARTS-like systematic sampling approach.

Although SMARTS achieves high simulation accuracy, its simulation speedup is still fundamentally constrained by its nature of uniform sampling as well as the speed of functional warm-up. On one hand, uniform instruction sampling treats all sampled units equally, and makes SMARTS unaware of the program phases. Since instruction samples from same program phase have similar behaviors, SMARTS may waste a large amount of time in simulating instruction samples that do not contribute to the simulation accuracy. On the other hand, functional warm-up, which is required to maintain the simulation accuracy, practically sets an upper bound for the potential speedup of SMARTS. To address the above limitations in SMARTS, we present a novel *Two Stage Sampling* (TSS) scheme. Unlike previous simulation sampling, this scheme samples instructions in two steps: in the first step, large groups of instructions (first stage samples) from the dynamic instruction stream are chosen as candidates for detail simulation; in the second step, small groups of instructions (second stage samples) from each of the first stage samples are chosen as the final points for detail simulation. Separating the sampling process into two stages allows us to monitor the statistics of each stage and remove redundant instruction samples, leading to a more efficient simulation. In particular, the contributions of this paper are as follows:

- **Two stage sampling framework:** This framework leverages two stage sampling to accelerate sampling-based simulation. Experimental results show that it can

This work is supported by National 973 Basic Research Program of China under grant No.2007CB310900

accelerate simulation speed while maintain high accuracy. Along with the framework, we also provide a set of mathematic analysis tool in determining the values of the tunable parameters.

- **Phase-aware sample reduction:** We propose a method to remove redundant instruction samples from detail simulation when the program is in stable program phase. It is unnecessary to do a complete and slow simulation for removing redundant samples. This sample pruning technique is especially useful in the iterative early design evaluation, where same program needs to run multiple times.

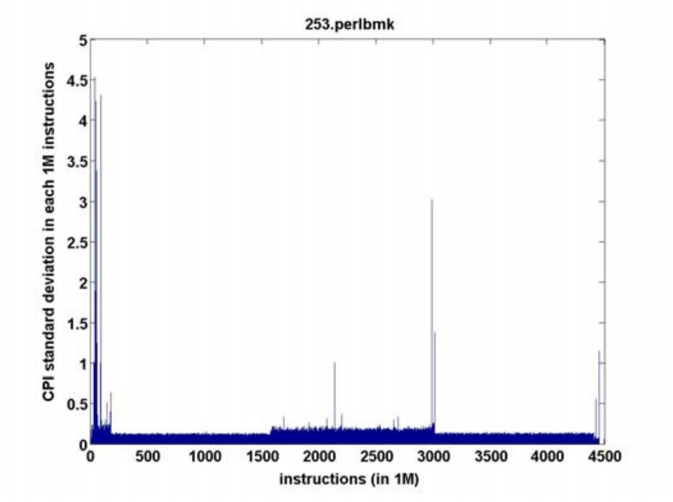
The organization of the paper is as follows: Section 2 describes the background and motivation. Section 3 shows the details of TSS approach. Section 4 provides the experiment setup, and the result analysis is presented in section 5. Section 6 gives the related work, and section 7 concludes the paper.

II. BACKGROUND AND MOTIVATION

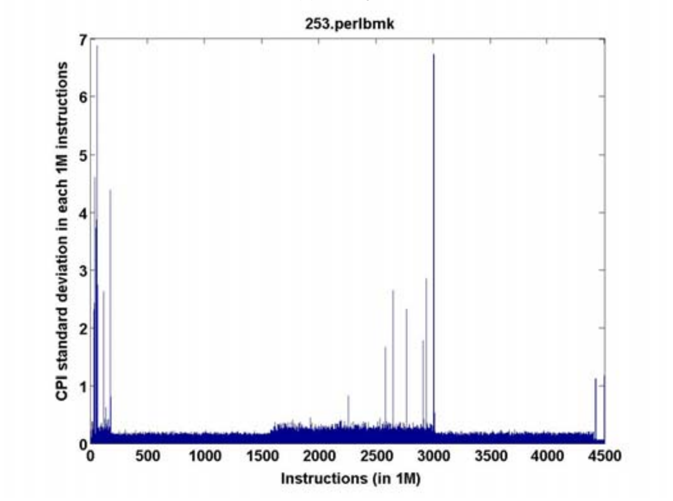
In SMARTS simulation framework, the original dynamic instruction stream of a program is broken into three kinds of non-overlapping chunks for functional warming, detail warming and detail simulation respectively. In the chunks for function warming, instructions are simulated on functional level with only large micro-architectural states, such as branch predictor and caches, maintained and updated. In the chunks for detail simulation, instructions are simulated at micro-architecture level with all relevant micro-architectural states updated in a cycle-accurate manner. The detail warming is essentially the same as detail simulation since it keeps track of all the micro-architectural states. It is introduced before running detail simulation in order to prevent the stale states in small micro-architecture units, like reservation station, from interfering the final performance estimates. All these chunks are distributed uniformly across the instruction stream.

This uniform sampling approach in SMART ignores an important program characteristic, that is, programs generally exhibit phase behavior [7]. Fig. 1 illustrates the program phases of *perlbmk* from SPEC CPU2000, running on two different processor configurations. Each bar in the figure represents the standard deviation of CPI in every 1 million instructions, with each CPI measured in an interval of 1000 instructions. When the bar is near 0, it means that CPI is similar within the 1 million instructions, and that detail simulation for one sampling unit is enough to estimate the overall CPI of the 1 million instructions. Therefore, by examining whether program is in stable phase, it is possible to further reduce the number of instructions for detail simulation without degrading the simulation accuracy. This is the key observation that motivates this study. In Fig.1, (a) and (b) have the similar shape of CPI standard deviation although the simulations run on totally different processor configurations. This is another observation which proves that program phases are mainly determined by program's inherent characteristics, and could be preserved across different micro-architecture configurations. This observation allows us to apply the phase information obtained from one simulation to the other simulations regardless of the micro-architecture changes.

In order to obtain the distribution of CPI standard deviation, small instruction sampling units should be grouped together to



(a) 8-way



(b) 16-way

Figure 1. CPI standard deviation distribution of *perlbmk* with different processor configurations. The hardware configurations of (a) and (b) are listed in Table II column 2 and column 3, respectively.

form a larger instruction chunk. Samples are selected from the larger chunk and measured. Consequently, the CPI standard deviation of the chunk could be calculated. In other words, the sampling process should be composed of two steps. Therefore, we employ two-stage sampling technique to exploit this opportunity.

III. TWO-STAGE SYSTEMATIC SAMPLING

This section presents the details of the framework for Two-Stage Sampling micro-architecture simulation (TSS). TSS is primarily developed around estimating average CPI, but it can also be used to estimate the power consumption.

A. *Technique overview*

Two stage sampling means the sampling process consists of two stages. Specifically, TSS first uniformly divides the dynamic instruction stream of a program into many large groups, i.e., first stage sampling units (*fsu*), and selects some of them as the candidates for detail simulation. In the second stage, TSS further divides every selected *fsu* into equally smaller groups, i.e., the second stage sampling units (*ssu*), and chooses some of them for detail simulation. The remaining unselected groups, including those from the first stages, are simulated in functional warm-up mode, which is used to ameliorate the cold start effect on the simulation accuracy. Fig. 2 shows how TSS alternates between functional warm-up and detail simulation in the dynamic instruction stream. The black bar represents detail simulation and the white bar represents functional warm-up.

To keep track of the program phase behaviors, TSS measures the CPI of every selected second stage sample. These obtained CPI measurements are used to calculate the CPI standard deviation of every selected first stage sample, which indicates whether the CPI in that sample changes smoothly or dramatically. If the CPI standard deviation is near zero, the variation of CPI in the *fsu* is very small and one *ssu* could represent the entire *fsu*. However, if the CPI standard deviation is far from zero, the CPI in the *fsu* has significant variation, and more *ssu* should be employed for detail simulation. In practice, we apply a threshold to the CPI standard deviation to differentiate these two cases. Since the CPI estimates are obtained when the program is simulated for the first time, the phase-aware sample reduction technique is not effective in the case that the program is simulated only once. However, in practice, especially in early design evaluation process, same programs are simulated multiple times to explore the performance of different design configurations. TSS could significantly accelerate these time-consuming iterative evaluation processes.

B. *Sampling unit sizes and strategy*

The implementation of TSS involves the sampling unit sizes as well as sampling strategy. The sampling unit size of first stage refers to the number of instructions in one first stage sample and determines the population size of the first stage sampling. Similarly, the sampling unit size of the second stage refers the number of instructions in one second stage sample and determines the population size in the second stage. In this study, we choose 1000 as the sampling unit size of the second stage because the coefficient of variation of CPI for SPEC2000 tends to level off when measurement unit size is larger than 1000 (instructions) [6]. The sampling unit size of the first stage affects the population sizes of both the first and second stages. In practice, its choice should ensure the population sizes in both first and second stages are large enough so that the statistical estimates are within high confidence level. In this study, we choose 1 million as the sampling unit size of the first stage.

After sampling unit sizes of the first and second stages are selected, the population sizes of them can be determined. The next step of implementation of TSS is to determine sampling strategy to select samples from the first and second populations. The sampling strategy affects simulation results and efficiency largely. After comparing simple random sampling, cluster sampling, strata sampling, and systematic sampling, we choose systematic sampling as the sampling strategies of the first and second stages for efficiency and accuracy. That is, for systematic sampling at an interval *i*, TSS repeatedly alternates between a functional simulation period of *i*-1 groups and detail simulation period of one group in order. The parameter *i* may be different for the first or second stage. In the next two sections, we will carefully discuss how to determine the sampling parameters of TSS.

C. *CPI estimation*

In the first stage, the dynamic instruction stream is divided into *fn* groups with the sample unit size *fs*. The relationship of between *fn* and *fs* is:

$$fn = \frac{L}{fs} \tag{1}$$

where *L* is the number of dynamic instructions of a benchmark. *fn* could also be written as:

$$fn = k \times kn \tag{2}$$

where *k* is the sampling interval (in terms of units) of the first stage, and *kn* is the number of the sampled units from *fn* groups. Similarly, in the second stage, we divide every selected sample from the first stage into *sn* groups:

$$sn = \frac{fs}{ss} \tag{3}$$

where *ss* is the sample unit size of second stages. *sn* could also be written as:

$$sn = k1 \times kn1 \tag{4}$$

where *k1* is the sampling interval of every second stage, and *kn1* is the number of groups selected from the *sn* groups.

For simplicity, we use y_{ij} to represent the CPI in the *j-th ssu* of the *i-th fsu* and $\bar{Y}i$ to represent the mean CPI of the *i-th fsu.* We have:

$$\bar{Y}_i = \frac{1}{sn}\sum_{j=1}^{sn} y_{ij} \tag{5}$$

The CPI standard deviation of the *i-th fsu* could be calculated via:

$$S_{2i}^2 = \frac{1}{sn-1}\sum_{j=1}^{sn}(y_{ij} - \bar{Y}_i)^2 \tag{6}$$

After sampling, we use *kn1* samples to the estimate of mean CPI of the *i-th fsu*:

$$\bar{y}_i = \frac{1}{kn1}\sum_{j=1}^{kn1} y_{ij} \tag{7}$$

The estimate of CPI standard deviation of the *i-th fsu* is:

$$s_{2i}^2 = \frac{1}{kn1-1}\sum_{j=1}^{kn1}(y_{ij} - \bar{y}_i)^2 \tag{8}$$

After getting the estimates of every selected first stage sample, we can calculate the estimation variables for the whole population by using the formula described in Table I.

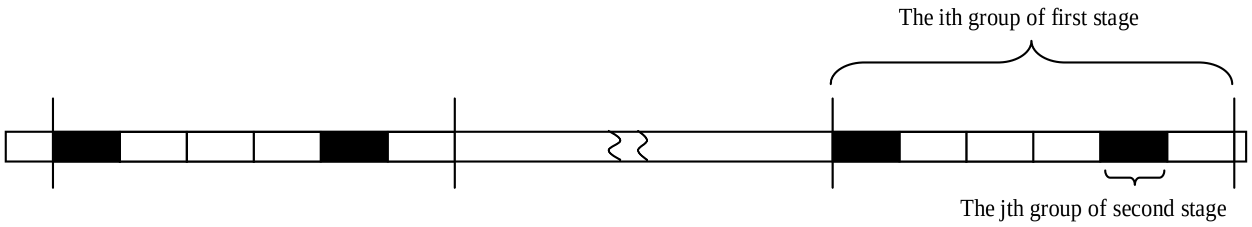


Figure 2 Two-stage sampling

TABLE I. SAMPLING VARIABLES OF TSS

Variables	Samples and estimates
nfsu	*kn*
nssu	*kn* × *kn1*
CPI1	$\bar{y} = \frac{1}{kn} \times \sum_{i=1}^{kn} \sum_{j=1}^{kn1} y_{ij}$
CPI2	$\bar{\bar{y}} = \frac{1}{kn \times kn1} \times \sum_{i=1}^{kn} \sum_{j=1}^{kn1} y_{ij}$
std1	$s_1 = \sqrt{\frac{1}{kn-1} \times \sum_{i=1}^{kn} (\bar{y}_i - \bar{\bar{y}})^2}$
std2	$s_2 = \sqrt{\frac{1}{kn \times (kn1-1)} \times \sum_{i=1}^{kn} \sum_{j=1}^{kn1} (y_{ij} - \bar{y}_i)^2}$

nfsu------number of sampled *fsu*
nssu------number of sampled *ssu*
CPI_1------CPI estimated by sampled *fsu*, represented by $\bar{y}$
CPI_2------CPI estimated by sampled *ssu*, represented by $\bar{\bar{y}}$
std_1------estimate of CPI standard deviation by sampled *fsu*, represented by s_1
std_2------estimate of CPI standard deviation by sampled *ssu*, represented by s_2

D. Sampling parameters

One of the most important parameters of TSS is the first stage sample size *kn*, which can be determined under a given confidence level and confidence interval of the CPI estimate. To calculate the confidence interval of CPI estimate, the mean-squared error of the CPI estimate should be calculated first. The unbiased estimation of mean-squared error is:

$$\hat{V}(\bar{\bar{y}}) = \frac{1-f_1}{kn} \times s_1^2 + \frac{f_1 \times (1-f_2)}{kn1 \times kn} \times s_2^2 \tag{9}$$

where $f_1 = \frac{kn}{fn}$, $f_2 = \frac{kn1}{sn}$

In practice, *fn*>>*kn* and *sn*>>*kn1*. So we assume $f_1 \approx 0$ and $f_2 \approx 0$. We have:

$$\hat{V}(\bar{\bar{y}}) = \frac{1}{kn} \times s_1^2 \tag{10}$$

Therefore, the coefficient of CPI variance can be estimated:

$$cv(\bar{\bar{y}}) = \frac{\sqrt{\hat{V}(\bar{\bar{y}})}}{\bar{\bar{y}}} \tag{11}$$

Given the confidence level 1-α, the maximum relative error of CPI estimate can be estimated with the following equation:

$$r = cv(\bar{\bar{y}}) \times u_{1-\frac{\alpha}{2}} \tag{12}$$

where $u_{1-\frac{\alpha}{2}}$ is the $1-\frac{\alpha}{2}$ quantile of normal distribution.

Therefore, the confidence interval of CPI estimate is:

$$d = r \times \bar{\bar{y}} = \frac{s_1 \times u_{1-\frac{\alpha}{2}}}{\sqrt{kn}} \tag{13}$$

To meet this given confidence level, *kn* must satisfy:

$$kn \geq \left(\frac{s_1 \times u_{1-\frac{\alpha}{2}}}{d}\right)^2 \tag{14}$$

To reduce the simulation time, the minimum integer value of *kn* should be chosen.

Theoretically, the initial values of the sampling parameters: *kn*, *kn1* could be set arbitrarily. After the trial simulation, CPI mean estimate, CPI mean variance estimate and the relative error can be obtained using the aforementioned formula. If the selected value of *kn* and *kn1* does not achieve a given confidence interval, one can calculate a new value to *kn* through the formula (14) and try again.

In practice, we could determine the initial value of *kn* through the following equation to accelerate the convergence:

$$kn_{ini} = \left(\frac{(s_1/\bar{\bar{y}}) \times u_{1-\frac{\alpha}{2}}}{r}\right)^2 \tag{15}$$

$(s_1/\bar{\bar{y}})$ is the variance of CPI and it is around 1 when sample unit size is larger than or equal to 1000 [10]. Hence, equation (15) could be simplified to:

$$kn_{ini} = \left(\frac{u_{1-\frac{\alpha}{2}}}{r}\right)^2 \tag{16}$$

According to the above equation, if one expects the relative error is within 3%, the kn_{ini} should be 4628 for confidence level of 95% and 10000 for 99.7%.

The value of the sampling parameter *kn1* should be at least 30 according to Center Limit Theory [8]. In this study, we set the value of *kn1* to 50.

E. Accelerating functional warm-up

In order to achieve high simulation accuracy, SMARTS employs functional warm-up to update the states of large micro-architecture units. Although functional warm-up (FW) is much faster than detail simulation, it still suffers from the slow speed compared with fast-forwarding (FFWD). As shown in Fig. 3, the average speed of FW is almost 3 times slower than FFWD on our experiment machine. In addition, the length of functional warm-up increases proportionally with the length of a program. Therefore, functional warm-up fundamentally limits the speedup of SMARTS. Although we also use functional warm-up to maintain simulation accuracy in TSS, we use it in a different way. That is, we substitute portions of functional warm-up with fast-forwarding. We call it systematic sampling functional warm-up (SSFW). The reason behind this is that large micro-achitecture units tend to use the recent history more often than the remote history due to program's temporal locality. Therefore, it is not necessary to update the states of cache and branch predictor all the way between detail simulation intervals.

Fig. 4 illustrates the TSS with systematic functional warm-up. Between the detail simulation intervals, some instruction groups are selected to simulate in functional warm-up mode and the other instructions are executed in fast-forward mode. The intervals between functional warm-up simulations are fixed. There are two parameters need to be determined: the functional warm-up length (*wl*) and the interval (*wk*). Since the state updates usually last a few thousands of instructions, we suggest setting *wl* to 10000. The choice of *wk* depends on not only the program inherent characteristics but also the size and structure of cache and branch predictors, which can also employ the procedure described in sub-section D of section III. In this paper, we set *wk* to 10 for simplicity.

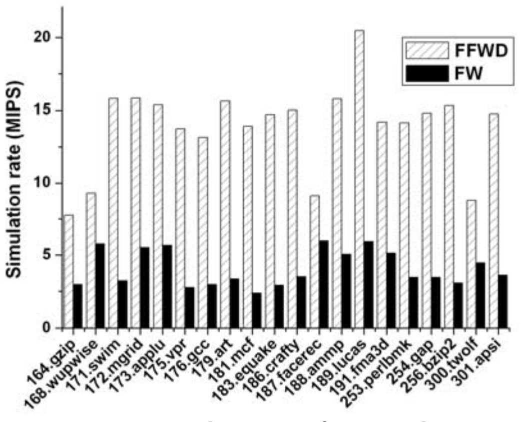


Figure 3 Simulation rates of FFWD and FW

IV. EXPERIMENT SETUP

To evaluate the performance of our two-stage systematic sampling approach, we use sim-outorder simulator in SimpleScalar tool set [9] to obtain the baseline simulation rate and accuracy. We also extensively modify SMARTsim [6] to support the proposed two stage sampling technique. The simulator models two different processors, the configurations of which are listed in Table II.

TABLE II. PROCESSOR CONFIGURATION

Parameter	8-way	16-way
Machine Width	8	16
Memory System	32KB 2-way L1 I/D 2 ports 8 MSHR 1M 4-way unified L2 16-entry store buffer	128KB 2-way L1 I/D 4 ports 8 MSHR 1M 4-way unified L2 16-entry store buffer
ITLB/DTLB	4-way 128 entries/4-way 256 entries 200 cycle miss	4-way 32 entries/4-way 32 entries 200 cycle miss
L1/L2/Mem latency	1/12/100 cycles	1/12/120 cycles
Function Units	4 I-ALU 2 I-MUL/DIV 2 FP-ALU 1 FP MUL/DIV	16 I-ALU 8 I-MUL/DIV 8 FP-ALU 4 FP MUL/DIV
Branch predictor	Combined 2K tables 7 cycle mispred 1 prediction/cycle	Combined 2K tables 7 cycle mispred 1 prediction/cycle

The workload of the experiment is composed of 21 SPEC CPU2000 benchmark programs listed in Table III. All of these programs are compiled to the alpha ISA. We use the reference input data set for every program throughout the experiments. According to the rules described in sub-section B of section III, we use 1 million as the first stage sample size. The initial values of *kn* and the corresponding *k* are shown in Table IV.

V. RESULTS AND ANALYSIS

In this section, we first evaluate the sampling-based functional warm-up strategy, followed by the evaluation of the TSS with phase-aware sample reduction techniques and sampling-based functional warm-up.

Fig. 5 shows the comparison of simulation rate between SMARTS and TSS with SSFW. The average speed of TSS with SSFW is 12.5 MIPS, in comparison to 10.78 MIPS by SMARTS, leading to a speedup of 1.16. Fig. 6 shows the comparison of the relative CPI error between TSS and TSS with SSFW. 8 out of the 21 benchmark programs have the same relative CPI error. 4 out of the 21 benchmark programs have a slight increase in relative CPI error. On average, the CPI relative error of TSS is 2.2% while that of TSS with SSFW is 2.24%. Therefore, by using sampling-based functional warm-up alone, TSS could achieve 1.16X speedup with negligible increase in error rate.

Since TSS has two sampling stages, it is easy to observe the CPI variation of first stage samples. If CPI is stable in a first

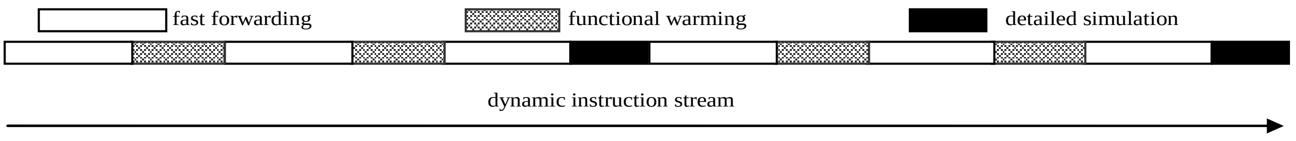


Figure 4. Systematic Sampling Functional Warm-up

stage sample, we can use only one second stage sample to represent the first stage sample without losing accuracy. We call this technique phase-aware sample reduction technique. When applying phase-aware sample reduction technique on top of TSS with SSFW, larger speedup could be obtained. As shown in Fig. 7, the combination of the two techniques in TSS yields speedup up to 2.29X with an average speedup 1.3X. Fig. 8 shows the CPI comparison among TSS with Program-phase-aware Sample Pruning techniques, SMARTS and the baseline

TABLE III. SPEC2000 BENCHMARKS

Benchmarks	Input data set	# of instructions (millions)
164.gzip	graphic	103,706
168.wupwise	wupwise.in	349,623
171.swim	swim.in	225,830
172.mgrid	mgrid.in	419,156
173.applu	applu.in	223,883
175.vpr	net.in, arch.in, place.in	67,724
176.gcc	166.i	46,917
179.art	c756hel.in	41,798
181.mcf	inp.in	61,867
183.equake	inp.in	131,518
186.crafty	crafty.in	191,883
187.facerec	ref.in	211,026
188.ammp	ammp.in	326,548
189.lucas	lucas2.in	142,398
191.fma3d	fma3d.in	268,368
253.perlbmk	diffmail.pl	39,925
254.gap	ref.in	269,037
255.vortex	lendian1.raw	118,972
256.bzip2	graphic	143,565
300.twolf	ref	346,484
301.apsi	--	347,923

TABLE IV. INITIAL PARAMETERS OF TSS FOR SPEC2000

Benchmarks	*kn*	*k*
164.gzip	4628	24
168.wupwise	4628	81
171.swim	4628	52
172.mgrid	4628	98
173.applu	4628	52
175.vpr	4628	15
176.gcc	4628	10
179.art	4628	9
181.mcf	4628	14
183.equake	4628	30
186.crafty	4628	44
187.facerec	4628	49
188.ammp	4628	76
189.lucas	4628	33
191.fma3d	4628	62
253.perlbmk	4628	9
254.gap	4628	63
255.vortex	4628	27
256.bzip2	4628	33
300.twolf	4628	81
301.apsi	4628	81

CPI (obtained by sim-outorder). On average, the relative error of CPI produced by TSS is 3.1% while that of SMARTS is 1%.

Generally, the CPI of a first stage sample is considered to be stable when CPI standard deviation is near zero. But what value of CPI standard deviation can be treated as near zero? To address this issue, we experimentally derive a threshold for the CPI standard deviation, below which the deviation can be regarded as negligible. Fig. 9 shows the CPI variation trend along with the CPI standard deviation threshold. As the threshold increases, fewer samples would be selected for detail simulation, which reduces the simulation time, yet compromises the simulation accuracy. Therefore, the value of the threshold should be carefully assigned to reach a desired trade-off between speedup and accuracy. We suggest that it can be set to 0.01 or 0.008 for most SPEC CPU2000 benchmarks.

Fig. 10 illustrates the performance of TSS with varied CPI standard deviation threshold. Generally, it is true that smaller CPI standard deviation thresholds require longer simulation time because more samples are simulated in detail. However, Fig. 10 shows that smaller CPI standard deviation thresholds may also lead to a shorter simulation time. This is because larger CPI standard deviation thresholds select fewer samples to simulate in detail and may result in more cache miss events. The miss penalty of these cache miss events may prolong the

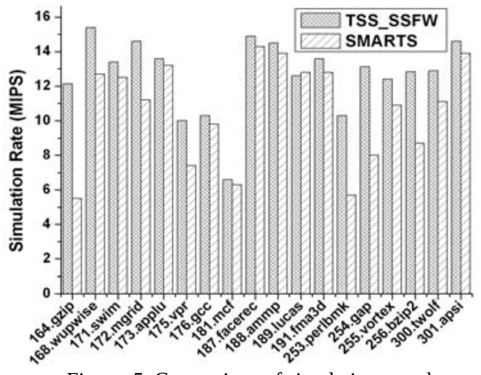


Figure 5. Comparison of simulation speed

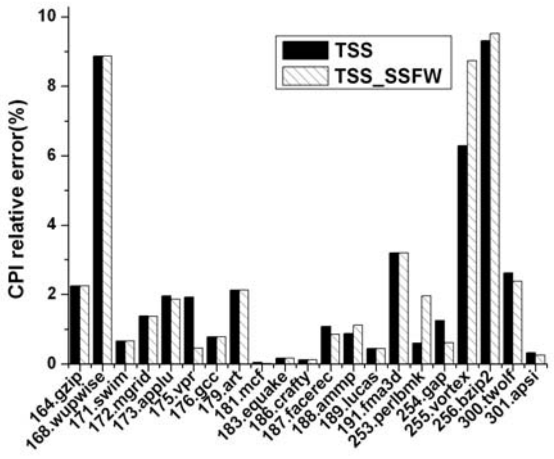


Figure 6. Comparison of CPI relative error

overall simulation time. This interesting observation implies that the threshold of CPI standard deviation should be chosen with cautions to reach a desirable simulation time and accuracy trade-off and it provides us an opportunity which can achieve high simulation precision with less simulation time. In this study, for most SPEC CPU2000 programs, it may result very high CPI precision with short simulation time when CPI standard deviation threshold is set to be less than 0.02.

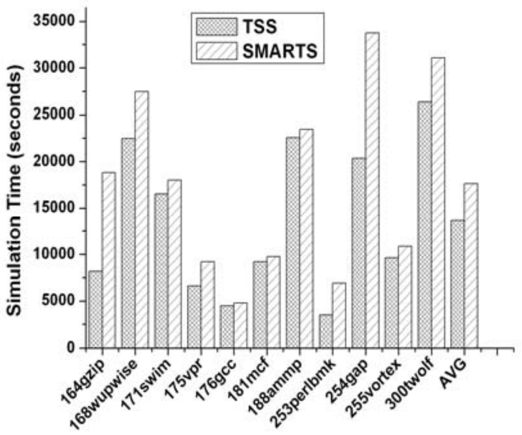


Figure 7. Simulation time of TSS with reduced samples and SMARTS. The standard deviation threshold is set to 0.01

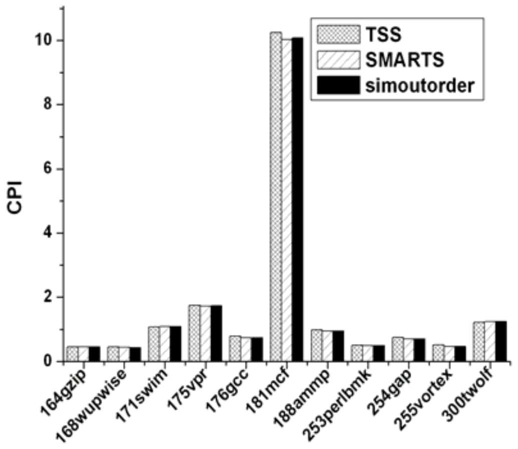
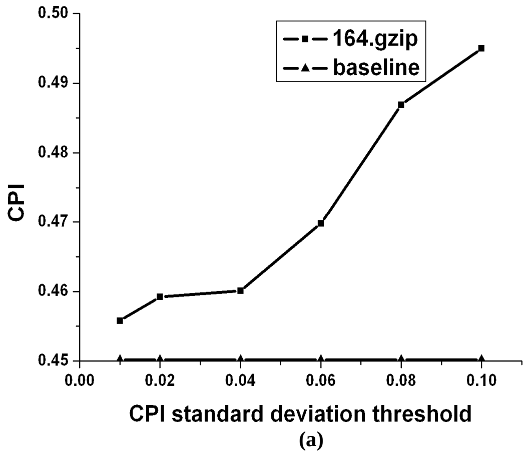
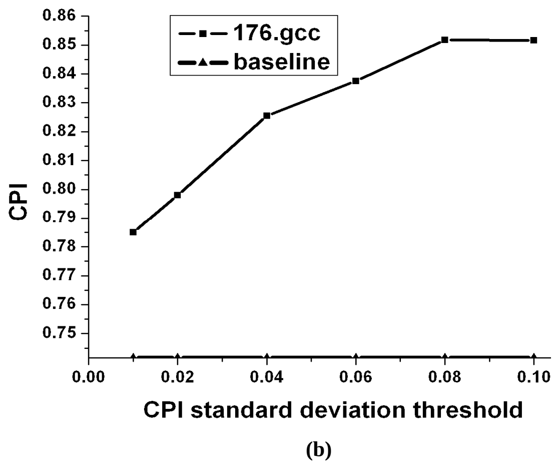
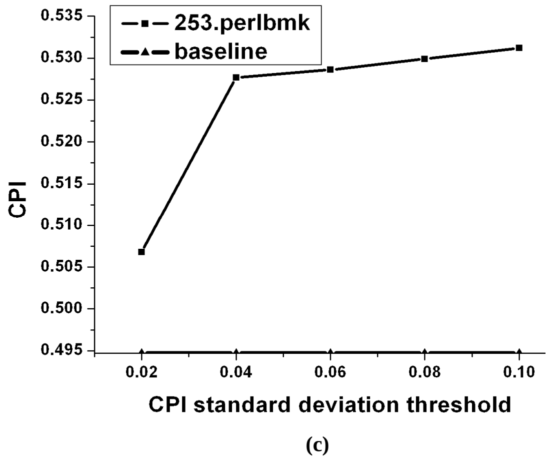


Figure 8. CPI of TSS with reduced samples and SMARTS. The standard deviation threshold is set to 0.01







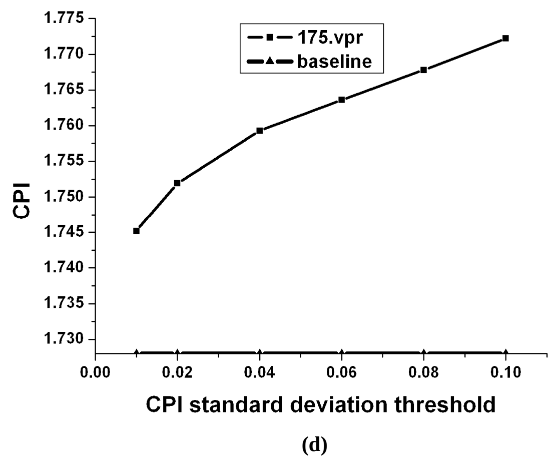


Figure 9 CPI variation trend along with the CPI standard deviation threshold. Because the space limit, we only show CPI trends for four benchmarks.

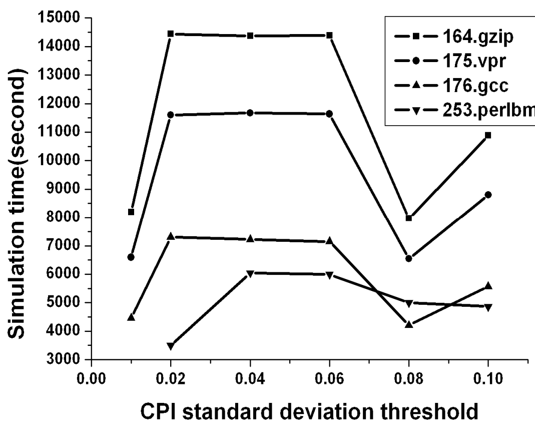


Figure 10 Simulation time variations with varied CPI standard deviation threshold.

VI. Related works

Many have tried to use several sampling techniques to select the instructions to execute on simulators in detail. The most natural sampling is the simple random sampling. However, it may generate poor accuracy results. To improve the accuracy of this technique, Conte et al. [10] suggests increasing the number of instructions dedicated to processor warm-up before each sample and/or increasing the number of samples. However, increasing the number of warm-up instructions or samples will increase the simulation time.

Representative sampling attempts to extract from a benchmark a subset of its dynamic instructions that matches its overall behavior when using the reference input set. Sherwood et al. [4][5][7][13] use BBV (basic block vectors) to characterize each instruction interval by profiling the benchmarks. They use *k*-means clustering to cluster the intervals based on their Euclidean distance and then select the interval closest to the centroid of each cluster. After each selected instruction interval executed on a simulator, they get the overall simulation results by summing the weighted individual results together. The experimental results of SimPoint (a simulator uses this technique) show that the simulation time can be dramatically decreased with an average IPC error of 3%. The main advantage of this technique is that it can be used across a wide range of micro-architectures because it analyzes micro-architecture independent characteristics of benchmarks and uses the results to select instructions for detail simulation. However, it may generate high simulation error and it does not provide a mechanism to assure accuracy of simulation results.

Wunderlich et al. [11] employ Stratified random sampling in their micro-architecture simulation study. They separate the distinct behaviors of a benchmark into different strata. Each behavior can be characterized by a small number of measurements. Each of these characterizations is then weighted by the size of the stratum to compute an overall estimate. Their results show that applying stratified sampling of SPEC CPU2000 benchmarks in simulation demonstrates an opportunity to reduce required measurement by 43 times over simple random sampling. Nevertheless, it is very difficult to separate benchmarks' full dynamic instruction stream into strata properly for simulation. Without a powerful stratification approaches, stratified sampling does not provide a clear advantage over simple random sampling.

Systematic sampling simulates selected portions of the dynamic instruction execution at fixed intervals. All intervals have the same length. The number of the intervals and the length of intervals determine the simulation time. SMARTS [6] implements this technique. In order to reduce the error caused by "cold-start", SMARTS adopts the functional warm-up. However, the disadvantages of SMARTS are two folds: (1) SMARTS ignores program-phases of programs, which may take time to simulation unnecessary instructions. (2) The functional warm-up of SMARTS limits its simulation speed. Even worse, functional warm-up requires simulation time proportional to benchmark length rather than sample size.

Since it is very difficult to determine the length of warm-up, many researchers have studied on it. Haskins et al.[16] measure the Memory Reference Reuse Latency (MRRL), which refers to the elapsed time measured in number of instructions between a reference to some memory address and the next reference to the same address. Instructions in a sampling unit and its pre-sample are profiled to get the distribution of MRRL. Since the distribution of the pre-sample may not be the same as that of the current sample, the warm-up length determined by this technique may not be accurate. Eeckhout et al propose the Boundary Line Reuse Latency (BLRL) method to avoid this problem [15]. However, neither MRRL nor BLRL takes the cache organization into consideration. Luo et al propose Self-Monitored Adaptive (SMA) warm-up method to consider the organization of caches and other factors synthetically [14]. Recently, Bryan et al [17] suggest reverse state construction reduce the length of functional warm-up. However, it is hard to apply these techniques to a wide range of sampling based simulation technologies.

VII. Conclusion

Micro-architecture simulation is an integrate part of modern processor design. However, it is difficult to achieve fast simulation rate with high accuracy. The new generation of SPEC CPU benchmarks and increasingly complicated processor designs exacerbate this problem. Even the state of art technologies such SMARTS and SimPoint are suffering simulation rate and accuracy challenges. This paper presents a new sampling scheme named two-stage sampling (TSS) and a novel warm-up strategy called systematic sampling functional warm-up (SSFW) to address this problem. TSS can apply the inherent program characteristics to reduce the samples simulated in detail. Our experiments show that TSS (with reduced sample technique and SSFW) achieve speed maximum and average speedup ratios of 2.29 and 1.3 over SMARTS while maintaining the same accuracy level. Additional, we find that the CPI standard deviation distribution pattern of benchmarks is insensitive to micro-architectures. Therefore, the information of one simulation trial can be used to accelerate the following simulation significantly.

REFERENCES

[1] A. J. KleinOsowski and D. J. Lilja, “MinneSPEC: A new SPEC benchmark workload for simulation-based architecture research”, *Computer Architecture Letters*, Vol.1, No.1, pp.7-11, January 2002

[2] A. Phansalkar, A. Joshi, and L. K. John, “Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite”, *Proceedings of the 34th annual International Symposium on Computer Architecture (ISCA’07)*, IEEE Computer Society, San Diego, California, USA, pp.412-423, 2007

[3] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John, “Measuring Benchmark Similarity Using Inherent Program Characteristics”, *IEEE Transactions on Computers*, Vol.55, No.6, pp.769-781, June 2006

[4] E. Perelman, G. Hamerly, M. V. Biesbrouck, T. Sherwood, and B. Calder, “Using SimPoint for Accurate and Efficient Simulation”, *Proceedings of the 2003 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, ACM Press, June 2003, San Diego, California, USA, pp.318-319

[5] T. Sherwood, E Perelman, G. Hamerly, and B. Calder, “Automatically Characterizing Large Scale Program Behavior”, *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM Press, October 5-9, 2002, San Jose, CA, pp.45-57

[6] R. E. Wunderlich, T. F. Wenisch, B. Fasafi, and J. C. Hoe, “SAMRTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling”, *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA’03)*, IEEE Computer Society, June 9-11, 2003, San Diego, USA, pp.84-95

[7] T. Sherwood, E. Perelman, and B. Calder, “Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications”, *Proceedings of the 2001 Conference on Parallel Architectures and Compilation Techniques*, IEEE Computer Society, September 10-12, 2001, Barcelona, Spain, pp.3-14

[8] Z. Govindarajulu, *Elements of Sampling Theory and Methods*, Pearson Education Asia Limited and China Machine Press, Beijing, 2005

[9] http://www.simplescalar.com/

[10] T. M. Conte, M. A. Hirsch, and K. N Menezes, “Reducing State Loss for Effective Trace Sampling of Superscalar Processors”, *Proceedings of International Conference on Computer Design*, IEEE Computer Society, October 7-9, 1996, Austin, Texas, USA, pp.468-477

[11] R. E. Wunderlich, T. F. Wenisch, B. Fasafi, and J. C. Hoe, “An Evaluation of Stratified Sampling of Microarchitecture Simulations”, *Proceedings of the Third Annual Workshop on Duplicating, Desconstructing, and Debunking*, IEEE Computer Society, June 19-23, 2004, Munchen, Germany, pp.13-18

[12] J. J. Yi, V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins, “Characterizing and Comparing Prevailing Simulation Techniques”, *Proceedings of the 11th international Symposium on High-Performance Computer Architecture (HPCA-11)*, IEEE Computer Society, February 12-16, 2005, San Francisco, pp.266-277

[13] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder, “Motivation for Variable Length Intervals and Hierarchical Phase Behavior”, *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS’05)*, IEEE Computer Society, March 20-22, 2005, Austin, Texas, USA, pp.135-146

[14] Y. Luo, L. K. John, and L. Eeckhout, “SMA: A Self-Monitored Adaptive Cache Warm-up Scheme for Microprocessor Simulation”, *International Journal of Parallel Programming*, Vol. 33, No.5, pp.561-581, 2005.

[15] L. Eeckhout, Y. Luo, K. De Bosschere, and L. K. John, “BLRL: Accurate and Efficient Warmup for Sampled Processor Simulation”, *The Computer Journal*, Vol.48, No.4, pp.451-459, 2005.

[16] J. W. Haskins and K. Skadron, “Memory Reference Reuse Latency: Accelerated Sampled Microarchitecture Simulation”, *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE Computer Society, 2003, Austin, Texas, USA, pp.195-203

[17] P. D. Bryan, M. C. Rosier, and T. M. Conte, “Reverse State Reconstruction for Sampled Microarchitecture Simulation”, *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE Computer Society, 2007, Austin, Texas, USA, pp.190-199