Using Statistical Theory to Study Issues in Microprocessor Simulation

Yue Luo and Lizy K. John
Department of Electrical and Computer Engineering
The University of Texas at Austin
luo@ece.utexas.edu   ljohn@ece.utexas.edu
IBM Technical Contact: Alex Mericas, *Processor Performance, Systems Group*

Abstract

Simulation of benchmarks has been the most important tool for computer architects to evaluate computer designs. In this paper, we employ statistical theory to study several key issues in microprocessor simulation. Sampling is an effective technique to reduce simulation time. Some approaches in the past use large number of small samples, while some other approaches use fewer numbers of larger samples. While fewer numbers of larger chunks is convenient, we study autocorrelation in programs and demonstrate that large chunks of continuous instructions do not capture additional information from the instruction stream. The high autocorrelation demonstrated by programs favors small sampling units.

Simulation is often used to evaluate the speedup of some microarchitectural enhancement. By applying ratio estimator in sampling theory we quantify the error of speedup measurements. Our result shows that speedup has smaller sampling error than CPI.

Next, we examine the validity of several reduced data sets for SPECint 2000. Using reduced data sets is an alternative to sampling, when one wants to reduce simulation time. Although the reduced data sets show CPIs vastly different from the reference data sets, the speedup from the reduced data set is not statistically different from the speedup from the reference data set. Confidence interval of speedup should be used as a guide to set target accuracy for designing new simulation techniques. Such future research needs to focus on more efficient and accurate warm-up mechanisms.

1. Introduction

Simulation of standard benchmarks has been the most popular method for computer architects to study the design tradeoffs. Modern benchmarks are no longer small kernels or synthesized toy programs. Instead, they are very close to real world programs and often take a long time to execute. Moreover, modern superscalar microprocessors are becoming increasingly complex; and so are the simulators modeling the processors. As a result, running benchmarks on detailed microarchitecture simulation models can take prohibitively large simulation times. Table 1 shows the time to simulate selected SPECint2000 benchmarks on a 1GHz Pentium III machine with sim-outorder, the detailed out-of-order superscalar simulator from the SimpleScalar 3.0 tool set [2]. It usually takes several days to simulate one program. Some benchmarks in the suite, which we could not afford to fully study, require weeks of simulation time.

Table 1. Number of instructions and simulation time of selected SPECint2000 benchmarks with reference data set. The data set name is integrated with the benchmark name.

Benchmark	Number of instructions (million)	Simulation time (days)
gcc-166	46,917	2.2
bzip2-source	108,878	4.4
eon-rushmeier	57,870	2.7
gzip-graphic	103,706	7.2
vortex-1	118,976	4.6
vpr-route	84,068	4.1
crafty	191,882	9.3

Since full simulation is impractical, a popular practice in computer architecture research is to fast forward billions of instructions to skip initialization phase of the program, and then simulate in detail several hundred million contiguous instructions. However, previous work has shown that this practice often results in large errors with respect to full simulation [12, 16]. To reduce simulation time yet retain good accuracy, researchers have employed many other techniques. Sampling and using reduced input sets are two of them.

Sampling techniques simulate multiple chunks of continuous instructions selected from the complete instruction stream. The instruction stream for simulation remains the same as the original, but only a small fraction of the original instruction stream is actually simulated in detail resulting in greatly reduced simulation times. A lot of work has been done in this area. However, much research work follows an ad-hoc approach: the newly proposed technique is evaluated solely experimentally in a few test cases to demonstrate its accuracy. Conte et. al. [4] applied sampling theory to processor simulation. They show how to calculate the confidence interval to quantify the error. Recently, Wunderlich et. al. [17] used sampling theory as guidance to design systematic sampling simulation. These statistical sampling approaches allow a confidence interval to be calculated to quantify the accuracy of the simulation without simulating the whole instruction stream.

MinneSPEC [11] is representative of the reduced input set approach. It consists of a set of reduced input sets to SPECcpu 2000 benchmarks. Therefore, when MinneSPEC is used, even though the benchmark programs remain the same, the dynamic instruction stream is smaller than that of the reference data set. KleinOsowski et. al [11] carefully profiled the SPEC programs and crafted the reduced input set so that MinneSPEC shows a program path profile very similar to that of the reference data set.

In this paper, we employ statistical sampling theory to study several important issues in microprocessor simulation. First, we examine the problem of sampling unit size. That is, we try to determine how large a chunk (the basic sampling unit) should be to achieve certain simulation accuracy while simulating as few instructions as possible. Some approaches in the past use large number of small samples [17], while some other approaches use fewer numbers of larger samples [16]. While fewer numbers of larger chunks is convenient, it is important to know whether increasing the chunk size really captures any more information than a smaller chunk. We study autocorrelation in programs to investigate the information content of the instruction stream and its variation with time. Our investigation indicates high autocorrelation in the studied programs, favoring small sampling units.

Another contribution of the paper is the use of ratio estimator to quantify accuracy of common metrics used in simulation. Traditionally, analysis of simulation methodologies has been based on achieving accurate CPI (Cycles Per Instruction). However, the goal of most simulation is to quantify the impact of some microarchitectural enhancement. Often, the final result of simulation is the speedup, not the absolute value of CPI. In section 3, we show how to use sampling theory to design an experiment to quantify the accuracy of speedup. We show that the sampling error in speedup estimation is significantly less than the error in CPI estimation.

Next, we compare different reduced input sets to answer the question: which data set is the most representative of the reference data set. We view SPECint 2000 as a sample from all CPU intensive integer programs. We observe that none of the reduced input sets are statistically representative enough for CPI estimations, but all of them are sufficiently representative for speedup estimates.

The view that a benchmark suite is just a sample from the set of programs that it represents has important implications. As discussed in Section 5, it provides target accuracies for designing new simulation methods.

2. Autocorrelation and its implication on sample size

Before going on, we first clarify some basic terminology because the same terms are often used differently in different research papers. We follow the established terminology in statistical sampling theory. The original full instruction stream is divided into N non-overlapping chunks of m continuous instructions. Each chunk is a basic simulation unit, or a *sampling unit*. The *sampling unit size* is the number of instructions in each chunk (m). The *population* refers to all the chunks that constitute the full instruction stream. *Population size* is the total number of sampling units in the full instruction stream, denoted N in this paper. A *sample* consists of selected chunks that are actually simulated and measured (In practice, more instructions are simulated for warming up microarchitecture). The number of sampling units in a sample is the *sample size*, expressed as n. The ratio of sample size and the population size is the *sampling fraction*, denoted by the letter f $(=n/N)$.

For convenience, systematic sampling is often used but is shown to be equivalent to simple random sampling in processor microarchitectural simulation [17]. The CPI of each sampled unit is measured (y_i , $i=1, .., n$). The CPI of the full simulation (population mean, $\bar{Y}$) is estimated as

$$\hat{\bar{Y}} = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \qquad \text{(eq 2.1)}$$

When normal distribution assumption applies, the confidence interval of $\hat{\bar{Y}}$ at confidence level (1-α) is

$$(\bar{y} - z_{1-\alpha/2}s_{\bar{y}}, \bar{y} + z_{1-\alpha/2}s_{\bar{y}}) \qquad \text{(eq 2.2)}$$

where $z_{1-\alpha/2}$ is the (1-α/2) quantile of a unit normal distribution, and $s_{\bar{y}}$ is the standard deviation of $\hat{\bar{Y}}$.

The final error of sampling simulation comes from two sources. The first source is the measurement error (i.e. inaccuracy in measuring the CPI of each sampling unit). To get the accurate CPI of a sampling unit, the state of all microarchitectural structures in the simulator must be correct at the beginning of the sampling unit. In practice, a number of instructions before the sampling unit are simulated to *warm up* the structure to obtain approximately correct microarchitectural states. Currently, the most accurate warm up is by functionally simulating caches and branch predictor throughout the full instruction stream [10, 17]. We assume that the measurement error is negligible in this section. The second type of error, which is the focus of this paper, comes from sampling itself, which is indicated by the variance $s_{\bar{y}}^2$. Similarly, we use the *coefficient of variation* (*cov*=$s_{\bar{y}}/\bar{y}$) to indicate the relative error. For example, following Equation 2.2 at confidence level of 99% the relative error of measured CPI is less than 2.58*cov*.

In sampling simulation of microprocessors, we face a unique question: we need to determine the number of instructions in a sampling unit (*m*). For example, if we have a budget of simulating 500 million instructions, to achieve a small error, should we simulate 5 chunks of 100 million instructions each, or 5000 chunks of 100,000 instructions each? And why?

In this study, we assume that the warm up overhead is constant. Although rarely true in practice, this assumption enables us to focus on the inherent property of the benchmark instruction stream instead of being tied down to a particular warm up scheme. We begin by studying the autocorrelation of the instruction stream. The CPI of each sampling unit is measured. The sequence of the CPIs, ordered in time, becomes a time series. *Autocorrelation* function of a time series $y_1, y_2, \ldots, y_N$ is defined as

$$\rho_h = \sum_{i=1}^{N-h}(y_i-\bar{y})(y_{i+h}-\bar{y}) \Big/ \sum_{i=1}^{N}(y_i-\bar{y})^2, \qquad \text{(eq 2.3)}$$

Autocorrelation is the correlation coefficient between neighbor sampling units with a lag (distance) of *h*.

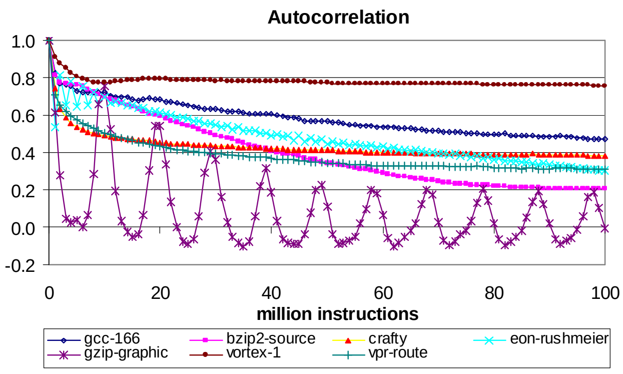


Figure 1a. Autocorrelation of CPI with 1 million

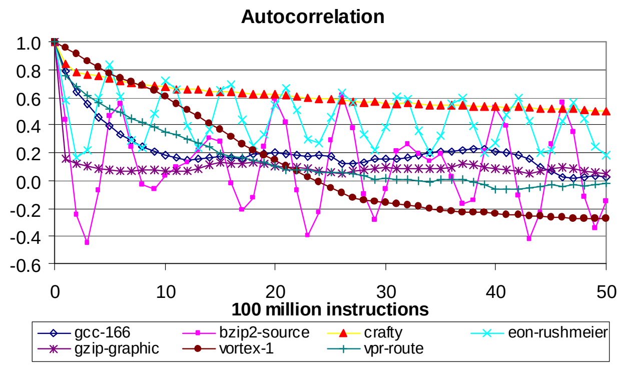


Figure 1b. Autocorrelation of CPI with 100 million instruction sampling unit

The full simulations of the 7 benchmarks in Table 1 are divided into sampling units. Figure 1 shows the autocorrelation of CPI of the sampling units for the benchmarks. The two sub-figures show the autocorrelation of different scale. In Figure 1a, each sampling unit is 1 million instructions, whereas the sampling unit size is 100 million instructions in Figure 1b.

It is clear that the autocorrelation curve is different for different benchmarks on different scales. Most benchmarks show a decreasing autocorrelation with increasing distances, but some exhibit periodicity in their CPI values. In this study we focus on the sign of the autocorrelation. Most benchmarks show high positive autocorrelation. A few benchmarks exhibit some negative autocorrelation values but the negative values have smaller magnitude (e.g. gzip-graphic in Figure 1a, bzip2-source in Figure 1b). At small lags all the benchmarks show high positive autocorrelation, which means that the CPI of one sampling unit is closely related to the units in its close neighborhood. The sign of autocorrelation especially at small lags determines the effectiveness of larger sampling units, as shown next.

As we have pointed out earlier, the standard deviation of the sample mean ($s_{\bar{y}}$) is used to evaluate the accuracy of the sample design. The smaller the standard deviation, the more accurate the sample result is. We use $s_{\bar{y}}(1)$ to denote our baseline standard deviation: sampling unit size of *m* instructions and sample size of *n* units. To evaluate the benefit of larger sampling unit, we compare two approaches of reducing the standard deviation. In the first approach, we choose a sampling unit size that is *j* times larger (*j*m* instructions) whiling keeping the sample size constant at *n*. Let $s_{\bar{y}}(j)$ denote the standard deviation in this case. In the second approach, the sampling unit size is not changed, but we take *j* times more units. $s'_{\bar{y}}(j)$ is used to denote the new standard deviation. Please note that the two approaches have the same number of measured instructions (*j*m*n*). It can be shown [1] that

$$s_{\bar{y}}(j) = \frac{s_{\bar{y}}(1)}{\sqrt{j}}B, \text{ where } B = \sqrt{1+2\sum_{h=1}^{j-1}(1-\frac{h}{j})\rho_h} \qquad \text{(eq 2.4)}$$

and

$$s'_{\bar{y}}(j) = \frac{s_{\bar{y}}(1)}{\sqrt{j}} \qquad \text{(eq 2.5)}$$

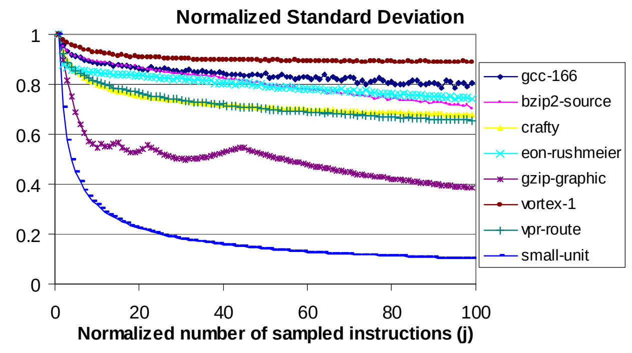
Equation 2.4 and 2.5 show that the benefit of increasing sampling unit size is decided by the autocorrelation ρ_h. If the neighbor sampling units are uncorrelated (ρ_h=0), then the two approaches give the same accuracy ($s_{\bar{y}}(j) = s'_{\bar{y}}(j)$). If neighbor sampling units show positive correlation (ρ_h>0), then increasing sample unit size is not efficient ($s_{\bar{y}}(j) > s'_{\bar{y}}(j)$). On the other hand, if neighbor sampling units are negatively correlated (ρ_h<0), then increasing sample unit size gives more accurate result than increasing sample size ($s_{\bar{y}}(j) < s'_{\bar{y}}(j)$). Please also note that the autocorrelation ρ_h with different lag *h* has different weight on the final result. The smaller the lag *h* is, the larger impact it has on the final result.

As shown in Figure 1, the autocorrelations are mostly positive especially when the lag is small, resulting in $s_{\bar{y}}(j) > s'_{\bar{y}}(j)$. Therefore, using larger sampling unit size is does not give good improvement in accuracy. To validate the conclusion, we calculate $s_{\bar{y}}(j)$ and $s'_{\bar{y}}(j)$ from their definition (not from Equation 2.4 and 2.5). The value normalized to $s_{\bar{y}}(1)$ is plotted in Figure 2. The base sampling unit is 1 million instructions in Figure 2a (m=10^6), and 100 million instructions in Figure 2b (m=10^8). The x-axis is j, the normalized number of measured instructions. Because different benchmarks show different autocorrelation, the $s_{\bar{y}}(j)$ curve is different for each benchmark. But all benchmarks share the same $s'_{\bar{y}}(j)$ curve (with legend "small-unit" in the figure). Figures 1 and 2 clearly reflect the relationship between autocorrelation and standard deviation of CPI as dictated by Equation 2.4. In Figure 1a, gcc-166 has the highest autocorrelation. As a result, it shows the largest standard deviation as the sampling unit size increases in Figure 2a. On the other hand, the autocorrelation of bzip2-source is the lowest in Figure 1a. Thus it exhibits the lowest standard deviation in Figure 2a for larger sampling unit. At the granularity of 100 million instructions, bzip2-source shows some negative autocorrelation in Figure 1b, so its standard deviation is very low in Figure 2b, even overlap the "small-unit" line for some points. This means that up to 600 millions, increasing the sampling unit size is as effective as taking more sampling units. Part of Vortex's autocorrelation is also negative, but the negative values come too late. The positive autocorrelation with the small lag has larger weight, resulting in relative high standard deviation for large sampling size in Figure 2b. As we can see from Figure 2, except for a couple of points in bzip2-source, larger sampling unit does not give as much error reduction in any of the benchmarks. Take crafty from Figure 2b as an example. Suppose we use a chunk of 100 million instructions as a sampling unit and the 95% confidence interval is e when simulating crafty. If we increase the chunk size to 1 billion instructions and keep the number of chunks the same, then we can only expect to limit the error to 0.89e, a marginal gain. However, if we keep the chunk size as 100 million instructions, but sampling 10 times more chunks, then our error limit is reduced to 0.32e, even though the total number of measured instructions stays the same as in the previous case.

To understand this result, we can consider correlation as similarity or predictability. To get good sampling accuracy, we want CPIs of our sample to cover as much as possible the CPI range of the population. However, if a sampling unit shows high correlation to its neighbor, then adding its neighbor to the simulation provides little additional information or coverage because the neighbor unit is very similar to the original sampling unit and its behavior is highly predictable with what we have already sampled. Therefore, simulating larger chunks of instructions is not effective at improving sampling accuracy.

One may expect the high autocorrelation to be the result of phase behavior. If a program exhibits phase behavior, a sampling unit will show very similar CPI to its neighbor units in the same phase. However, our experiment shows that high autocorrelation is more universal than phase behavior. Figure 3 shows the CPI of every 100 million instruction sampling unit for two benchmarks: vortex-1 and crafty. Vortex-1 has been the subject of many phase behavior researches, whereas the CPI graph of crafty is close to noise to human's eyes. However, both benchmarks exhibit high autocorrelation as shown in Figure 1b.

We believe that the underlying reason for high autocorrelation is temporal locality. Because of temporal locality, a sampling unit executes similar code and accesses similar data as its close neighbor, which results in very similar CPI as its neighbor. Temporal locality has been proven to be a basic behavior of all programs and it exists on a wide range of scales (e.g. microprocessor cache hierarchy). Therefore, it is no surprise to see this universal non-negligible autocorrelation in the instruction stream simulation.



(a) Base unit size is 1 million instructions

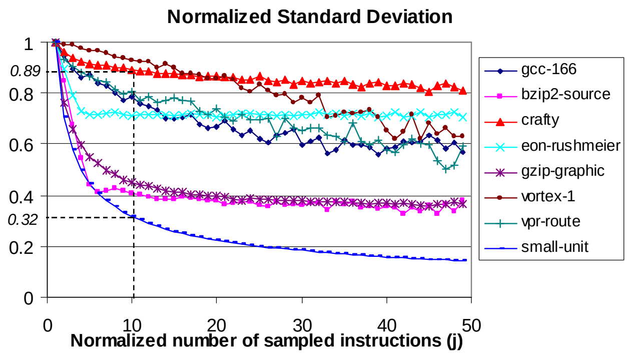


Figure 2. Normalized standard deviation CPI for different sampling unit sizes

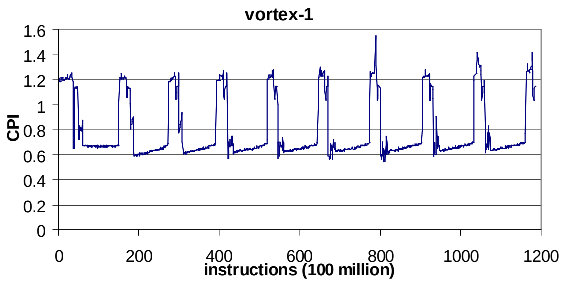


Figure 3a. CPI of every 100 million instruction unit for vortex-1

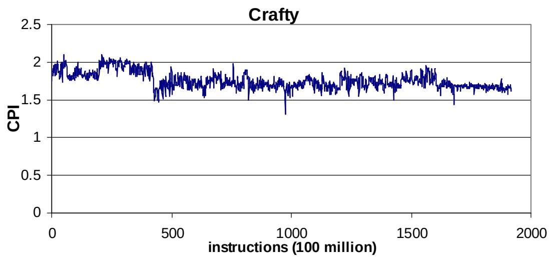


Figure 3b. CPI of every 100 million instruction unit for crafty

In summary, the instruction stream shows high positive autocorrelation between close neighbor sampling units. We have demonstrated mathematically that the high autocorrelation favors small sampling unit. Our result shows that simulating larger and larger continuous chunks of instructions is not an effective way to improve accuracy. Because of the ubiquity of temporal locality, this conclusion is expected to hold true for almost all programs.

A word of caveat may be in order here. In the above discussion, we assume that the overhead of warm-up is constant. In reality, the warm-up overhead usually increases if we reduce the sampling unit size and increase the sample size. As such, the sampling unit size should be a trade-off between accuracy and the simulation overhead depending on the specific warm-up scheme.

3. SAMPLING ERRORS IN SPEEDUP AND CPI

Previous research on sampled simulation of microprosessor generally focuses on the accuracy of CPI or IPC. However, the goal of a simulation is usually to evaluate the benefit of some architectural enhancement, in which case, the absolute value of CPI may not be overly important. Instead, an accurate estimate of the speedup is often a more desired metric. We define the speedup R as the ratio of the CPI before the enhancement to CPI after the enhancement when the same benchmark is run. There is nothing wrong with pursuing accurate CPI value because more accurate estimation of CPI will naturally result in better accuracy of speedup. However, the accuracy for the two metrics show different properties as we shall see later.

We employ the ratio estimator in sampling theory to calculate the speedup and to quantify its error. For each sampling unit, there are two characteristics, y_i and x_i (i=1, 2, ..., N). We randomly take a sample of size n and measure y_i and x_i of each sampled unit (i=1, 2, ..., n). Our goal is to estimate R, the ratio of the population mean of y to the population mean of x ($R = \bar{Y} / \bar{X} = \sum_{i=1}^{N} y_i \Big/ \sum_{i=1}^{N} x_i$).

Based on sampling theory, R is estimated as

$$R = \frac{\bar{y}}{\bar{x}} = \sum_{i=1}^{n} y_i \Big/ \sum_{i=1}^{n} x_i \qquad \text{(eq 3.1)}$$

Its variance is estimated as

$$v(\hat{R}) = \frac{(1-f)}{n\bar{x}^2}(s_y^2 + \hat{R}^2 s_x^2 - 2\hat{R}s_{yx}), \qquad \text{(eq 3.2)}$$

where $$s_{yx} = \frac{\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})}{n-1} \qquad \text{(eq 3.3)}$$

If the sample is large enough so that the normal approximation applies, the confidence interval for R can be obtained as

$$(\hat{R} - z_{1-\alpha/2}\sqrt{v(\hat{R})}, \hat{R} + z_{1-\alpha/2}\sqrt{v(\hat{R})}). \qquad \text{(eq 3.4)}$$

Based on the above theory, we propose the following steps to calculate the speedup and quantify its error.

1. Divide the full instruction stream into N chunks of m continuous instructions. Take a systematic sample or random sample of size n.
2. Measure the CPI of each sampled unit before the architectural enhancement. Record all the CPIs (x_i).
3. Measure the CPI of the *same* sampled units after the enhancement. Record all the CPIs (y_i).
4. Calculate the speedup, its standard deviation and confidence interval with equations 3.1 through 3.4.

The key point is to make sure the same sampled units are measured in the two simulation steps. Two problems can potentially prevent us from achieving this. Firstly, the instruction stream may be different in each run of the same benchmark. For a user mode simulator like SimpleScalar, this is caused by operating system calls (e.g gettimeofday) returning different result in each run. For example, in two runs of gcc-166, the difference in the number of dynamic instructions was 332,372. Although this difference only accounted for 0.00071% of the total instructions executed, it would cause different units to be sampled in the two runs because of the small sampling unit size (1,000 – 10,000 instructions). To solve this problem, one must make sure that the dynamic instruction stream in each run is exactly the same. In our experiment, we first capture the *eio* trace with SimpleScalar sim-eio utility. Then all the benchmark programs are run with the eio trace to guarantee the same instruction sequence. Secondly, the architecture simulation events are aligned with clock cycles, not instructions. This can cause problem for simulating superscalar processors, which are capable of committing multiple instructions in a single clock cycle. Suppose that one sample unit is from instruction #100 to #199. In the first simulation, instructions #98-#101 are committed in the same cycle. In the second simulation with the microarchitectural enhancement, instructions #99-#103 are committed in one cycle. Obviously, the sampling units cannot be exactly the same in the two runs if we count whole cycles. There are two ways to solve the partial cycle problem. In the first solution, if *i* instructions are committed in one cycle, we (artificially) allocate 1/*i* cycle to every instruction in this cycle. In the above example, instruction #100 and #101 in the first run will be counted as 2/4=0.5 cycles. This approach strictly meets the ratio estimation requirement but requires some additional book keeping. The other simpler approach, which we opted for in our experiment, uses larger sample unit size. We start and stop measuring at the boundary of clock cycles, but because of the large sample unit (10,000 instructions) the misalignment of the sample units in the two runs is negligible.

We conducted an experiment to show the validity of applying ratio estimation theory to sampled simulation. Eight benchmarks from SPECcpu 2000 are simulated in a modified SimpleScalar 3.0 sim-outorder simulator, which performs the above systematic sampling procedures. Each sampling unit is 10,000 instructions and 3,000 units in every benchmark are simulated. Caches and branch predictors are continuously warmed up functionally as in [10, 17]. 4,000 instructions before every sampling unit are simulated with cycle accurate simulator to warm up other microarchitecture structure. An 8-way and a 16-way out-of-order superscalar processor are simulated to calculate the speedup. The microarchitecture configurations are given in Table 2 [17].

The results are shown in Table 3. Table 3a shows the CPI result for the 8-way configuration. "Sampling result" column shows the CPI computed by sampling whereas "True value" is the CPI from the full simulation by sim-outorder. The actual relative error is shown in the last column. The estimated coefficient of variation (COV) is shown in column 3. Table 3b shows the result for 16-way configuration in the same format. The results for speedup of 16-way machine vs 8-way machine are shown in Table 3c. The second column is the speedup calculated from Equation 3.1 whereas the "True value" is calculated as the ratio of the true CPIs of the two configurations. COV for speedup is shown in column 3.

First, we examine the coefficient of variation, which indicates the error solely due to sampling. In all benchmarks the speedup invariably shows smaller COV than the CPI. The resultant benefit is that to achieve a specific limit of relative error, even fewer sampling units need to be measured than in CPI. Suppose that we want the relative error to be within 2% at the confidence level of 95%. The required sample size can be calculated from Equation 2.2 for CPI and Equation 3.4 for speedup. The result is shown in Figure 4. Though required sample size varies greatly from benchmark to benchmark, the sample size for the speedup is only a small fraction of that for CPI. It will take fewer simulated instructions to achieve the same accuracy for speedup than for absolute CPI value.

Table 2. Processor configurations

Parameter	8-way (baseline)	16-way
Machine Width	8	16
RUU/LSQ size	128/64	256/128
Memory System	32KB 2-way L1 I & D, 2 ports, Unified 1M 4-way L2	64KB 2-way L1 I & D, 4 ports, Unified 2M 8-way L2
ITLB / DTLB	4-way 128 entries 4-way 256 entries 200 cycle miss penalty	4-way 128 entries 4-way 256 entries 200 cycle miss penalty
L1/L2/Memory Latency	1/12/100 cycles	1/16/100 cycles
Functional Units	4 I-ALU 2 I-MUL/DIV 2 FP-ALU 1 FP-MUL/DIV	16 I-ALU 8 I-MUL/DIV 16 FP-ALU 4 FP-MUL/DIV
Branch Predictor	Combined 2K tables 7 cycle misprediction penalty 1 prediction/cycle	Combined 8K tables 10 cycle misprediction penalty 2 predictions/cycle

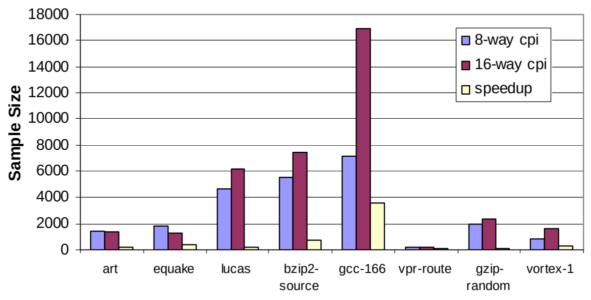


Figure 4. Sample size required to achieve 2% relative error at 95% confidence level

This conclusion may seem counter-intuitive at first. The speedup is calculated as the ratio of two CPIs as in Equation 3.1. Then how could the speedup be more accurate than each CPI? The answer lies in the fact that different parts of the benchmark program usually benefit similarly from the microarchitectural enhancement even though the absolute value of CPI may vary widely during the execution. COV is an indicator of the degree of variation in the population. The value of speedup is more uniform among the sampling units than the value of CPI, resulting in a smaller COV, and a tighter confidence interval for the speedup.

Next, we look at the actual error, which consists of the sampling error discussed above and the error due to inaccuracy in the measurement of the CPI of each sample unit. The latter is mostly caused by inadequate warm-up of large microarchitecture structure. The most accurate warm-up scheme [10, 17] in the literature is used in our experiment. The caches and the branch predictor are warmed up throughout the full simulation, but they are only functionally simulated for most of the time. The speculative behavior is not modeled except for during the units that are simulated in detail. In some benchmarks, this measurement error becomes dominant. For example, given a sample size of 3000, the COV for vpr-route is so small that at 99% confidence level, the relative error of the CPI on the 16-way machine should be within 1.91%. However, the actual error is about 4%, so it is almost certain that the final error mainly consists of the measurement error. The error due solely to sampling decreases quickly when we increase the sample size. If we can accurately measure just 30 million instructions (0.02%-0.06% of total dynamic instructions), the error in speedup will be below 1.5% for all benchmarks except gcc. However, in reality the measurement error due to imperfect warm-up quickly becomes a limiting factor on the accuracy. Therefore, future research on new simulation methodology needs to focus on more efficient and accurate warm-up techniques.

Table 3a. CPI for 8-way configuration

Benchmark	Sampling result	COV (%)	True value	Relative error (%)
art	1.0451	0.7112	1.0442	0.0869
equake	1.3438	0.7937	1.3448	0.0707
lucas	2.4237	1.2774	2.4931	2.7825
bzip2-source	0.6003	1.3915	0.5959	0.7322
gcc-166	0.5057	1.5814	0.5042	0.3037
vpr-route	1.5603	0.2233	1.5410	1.2502
gzip-random	0.4243	0.8194	0.4224	0.4608
vortex-1	1.0451	0.7112	1.0442	0.0869

Table 3b. CPI for 16-way configuration

Benchmark	Sampling result	COV (%)	True value	Relative error (%)
art	0.5866	0.6810	0.5923	0.9586
equake	0.9309	0.6480	0.9318	0.1006
lucas	2.2679	1.4612	2.3525	3.5973
bzip2-source	0.4866	1.6046	0.4791	1.5581
gcc-166	0.2816	2.4210	0.2714	3.7609
vpr-route	1.3519	0.2430	1.3301	1.6409
gzip-random	0.3500	0.8923	0.3458	1.2279
vortex-1	0.3235	0.7408	0.3103	4.2485

Table 3c. Speedup (16-way vs 8-way)

Benchmark	Sampling result	COV (%)	True value	Relative error (%)
art	1.7816	0.2770	1.7630	1.0556
equake	1.4437	0.3558	1.4432	0.0300
lucas	1.0687	0.2745	1.0598	0.8452
bzip2-source	1.2337	0.4950	1.2438	0.8132
gcc-166	1.7959	1.1212	1.8578	3.3318
vpr-route	1.1541	0.1300	1.1586	0.3844
gzip-random	1.2123	0.1488	1.2215	0.7577
vortex-1	1.5689	0.2807	1.5962	1.7093

4. Comparing reduced data sets

SPECcpu 2000 comes with three data sets: reference, train, and test. Only the reference data set is supposed to be used to evaluate computer performance. However, the reference data set takes such a long time to run that it is impractical to use it to evaluate multiple microarchitectural alternatives by simulation. Besides sampling, reduced data set is another approach to reduce simulation time. The same program is executed and cycle accurate simulation is done throughout the whole execution. But the input data set to the benchmark program is reduced resulting in much shorter simulation time. Reduced data sets for SPECint 2000 include train, test and MinneSPEC. Test is not intended to perform any simulations, while MinneSPEC is a small data set specifically designed for microprocessor simulation. Previous research evaluate the reduced data sets by comparing one program by one program the absolute value of CPI and other microarchitecture metrics between the reduced data set and the reference data set [5, 7].

In this section, we take a different approach to evaluate the reduced data sets. Firstly, we recognize that the goal of a simulation experiment is to assess some architectural improvement. Therefore, the accuracy in speedup is often more important than the accuracy in absolute CPI value. We will compare both CPI value and speedup. Secondly, we do not base our conclusion on one to one comparison of benchmarks or on the “average” error. Instead, we view SPECint 2000 as a sample from all the CPU intensive integer programs in the world (the population). We assume that SPECint 2000 is a simple random sample. (Unless there is some randomness in the sampling, no statistical theory can be developed for the approach and no statistical conclusion can be made about the population). By comparing the reduced data sets, we are trying to find out if they represent the same “population” as the reference data set does. As long as the populations are the same, the reduced data sets are equivalent to the reference data set when used to evaluate the performance improvement of computer designs.

If the populations are the same, then the median of the population should be the same. This question is best answered by hypothesis testing. Because we do not know the distribution of the CPI, we choose Wilcoxon signed rank test, which does not assume normal distribution, to test the equality of the population median. The test requires that the sampling units be independent of each other. However, in some data set, one benchmark program is run with several input sets. For example, the bzip2 program has 3 input sets in the reference data set. Previous research [5] has shown that the performance metrics for these input sets to the same benchmark program may be quite similar; thus their CPI and speedup are not independent of each other. If a program takes multiple input sets within one data set, we calculate the arithmetic mean as the value for this program. As we cannot afford to fully simulate the reference data set in sim-outorder, we use the sampling method in the previous section to gather the speedup and CPI for reference data set. The sampling method will incur some small errors, but they are negligible compared to most errors in the reduced data set. The relative errors in CPI and speedup of the different reduced data sets are plotted in Figure 5[1]. Some benchmarks show very large errors when reduced input sets are used. Again we see the repeated pattern: the error in speedup is often much smaller than the error in CPI. It is also interesting to note that the average error of test, train and MinneSPEC data set are close, with MinneSPEC showing an edge in the speedup estimation. Then we use Wilcoxon signed rank test to test if the median of the two populations are the same. Each reduced data set (test, train and MinneSPEC) is tested against the reference dataset in terms of CPI on 8-way machine, CPI on 16-way machine, and the speedup. The Wilcoxon test result is summarized in Table 4. We choose a significance level of 0.05. Except for the test data set on the 8-way machine, which barely passes the test at this significance level, none of the reduced data sets has the same median CPI as the reference data set. However, all the *p*-values for the speedup are above the significance level. Therefore, using the reduced data set to evaluate the speedup in our experiment is not statistically different from using the reference data set.

By comparing different data set, we are not only interested in the population mean, but also we want to see whether the populations follow the same distribution. If the population distribution is the same, then the distribution of the sample mean will also be the same. To visually show the distribution of sample mean, we employ bootstrapping [6], a modern computer-simulated, nonparametric technique to statistical inference. In our experiment, we draw 10,000 resamples. The histograms of the CPI and speedup are shown in Figure 6. The x-axis is deliberately drawn on the same scale for easy comparison. It is obvious that the distribution of the sample mean CPI for reference data set is far from normal. Furthermore, different data set shows vastly different distributions. The multiple peaks in sample mean CPI distribution of reference data set are the result of several programs (notably, mcf) showing distinctively higher CPI than others. This property is not retained in test and MinneSPEC, where the CPIs of all the benchmarks are closer to each other resulting in narrower and single-peak distribution. However, for the more important metrics, the speedup, the distributions of different data set are more similar to each other. In addition, the distribution of the sample mean of the speedup also looks more like normal distribution. A quantile-quantile plot of the reference speedup against normal distribution (Figure 7) shows that the distribution of the speedup of reference data set is fairly close to the normal distribution (but with slightly shorter tails).

[1] We were not able to run perlbmk or parser in the simulator, so we have results for out of 12 SPECint benchmarks. For better statistical result, more benchmarks are needed.

Table 4 Wilcoxon signed rank test of different reduced data sets

Metrics	Reduced data set	*p*-value
CPI on 8-way	Test	0.06445
	Train	0.02734
	MinneSPEC	0.04883
CPI on 16-way machine	Test	0.03711
	Train	0.01953
	MinneSPEC	0.03711
Speedup (16-way vs. 8-way)	Test	0.999
	Train	0.375
	MinneSPEC	0.6953

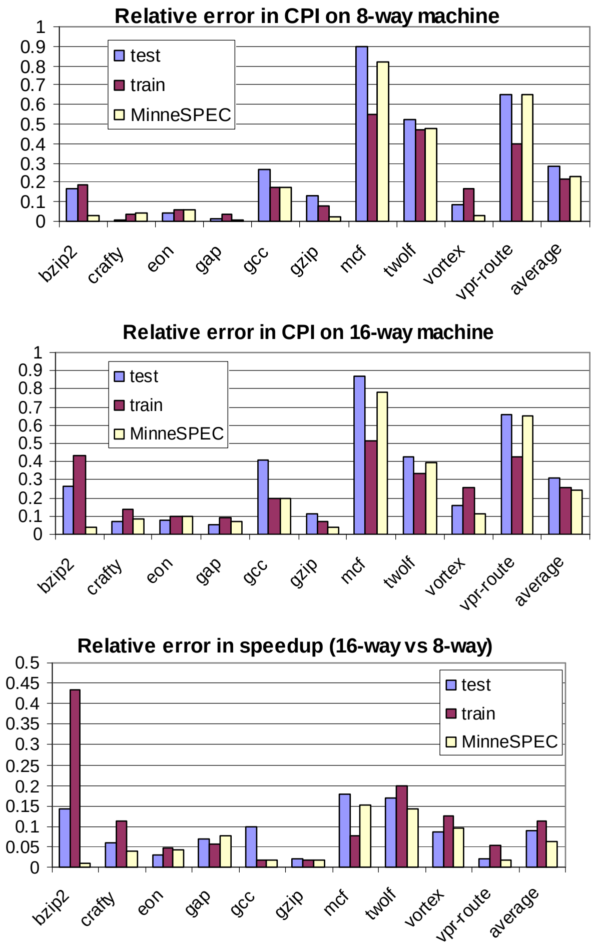


Figure 5. Relative errors of different reduced data sets

In summary, none of the reduced data set (test, train, MinneSPEC) represents the reference data set in terms of CPI. However, one can use any of the reduced data set to evaluate the speedup and draw the statistically same conclusion about the performance of the processor. This interesting observation is currently based only on our experiment of two processor configurations. Although we expect that the reduced data set and reference data set will show more similarity in speedup than in CPI, more experiments are needed to test its general applicability.

5. Confidence interval of speedup

Computer designers run benchmarks to evaluate design alternatives, but no user runs these benchmarks in their everyday work. Therefore, the real question a computer designer is trying to answer is: how well will the computer design perform for all CPU intensive workload in the world based on the result of the SPECcpu suite? To answer the question, we view SPECint as a random sample from all CPU-intensive integer programs and calculate the confidence interval for the mean speedup. The previous section has shown that the mean speedup approximately follows the normal distribution. Therefore, Equation 2.2 can be used to calculate the confidence interval. Another method is to use bias corrected bootstrapping [6], which does not rely on the assumption of a particular underlying distribution. The results are shown in Table 5. The last column is the limit of relative error converted from the confidence interval. We can see that bias corrected bootstrapping often results in tighter interval.

The confidence interval can serve as guidance for target accuracy when the computer architecture researcher designs future simulation techniques. If the limit of relative error is 8%, then the error in simulating each benchmark programs should be much smaller than 8% (e.g 1%). On the other hand, we should not shoot for unnecessary accuracy such as an error of 0.1%, which will be wasteful of simulation resources. The current sampling simulation gives an error below 4% (Table 3). It is close to meet the requirement but smaller errors are still desirable. Furthermore, most of today's microarchitectural enhancement in literature does not offer a speedup as large as the difference between a 16-way processor and an 8-way processor. Therefore, the confidence interval for the whole benchmark suite will be tighter and even smaller errors in each benchmark program simulation are required.

6. Related work

Wunderlich et. al. [17] proposed using multiple small sampling units to get accurate simulation CPI. They employed sampling theory to calculate the confidence interval and to select the sample size at a given accuracy requirement. SimPoint is another recently proposed sampling simulation scheme. It uses cluster analysis based on basic block vector to select representative simulation chunks. The latest version allows the user to quantify the error in CPI with a confidence interval on the original architecture for which the full simulation was done. Our work focuses on measuring speedup instead of CPI, which is more important to computer designers. More sophisticated sampling theory is employed to calculate the confidence interval of the speedup. We show that at the same accuracy level, speedup requires smaller sample size than CPI, so it is more efficient to estimate the confidence interval directly with our method. Wunderlich et. al. also showed that smaller sampling unit is more effective than large sampling unit that had been commonly used in previous research. The evolution of SimPoint also exhibits the trend of smaller sampling units. The precursor of SimPoint [15] simulated a large chunk (300 million) of instructions. The original SimPoint [16] used several 100 million instruction chunks. In Variance Simpoint [14], the latest version of SimPoint, on average about 100 chunks of 1 million instructions are simulated. We confirmed Wunderlich et. al's conclusion but we have further explored the underlying reason by studying the autocorrelation of the instruction stream. We show that this phenomenon is caused by high autocorrelation inside the instruction stream, which is an expected result of temporal locality.

Table 5. The confidence interval of speedup on a 16-way processor vs 8-way processor

Confidence level	Estimation method	Confidence interval	Equivalent relative error limit
95%	Normal distribution	(1.214, 1.441)	8.5%
	Bias Corrected	(1.232, 1.449)	8.2%
99%	Normal distribution	(1.179, 1.476)	11%
	Bias Corrected	(1.208, 1.485)	10%

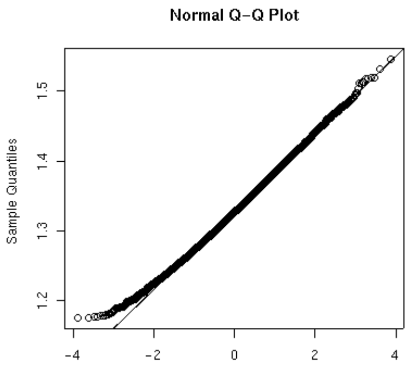


Figure 7. Quantile-quantile plot of the speedup of reference data set against normal distribution. The line, which passes through the 1st and 3rd quartiles, is for comparison.

Hsu et. al. [8] compared the IPC and path profile of test, train, and reference data set. They studied how the

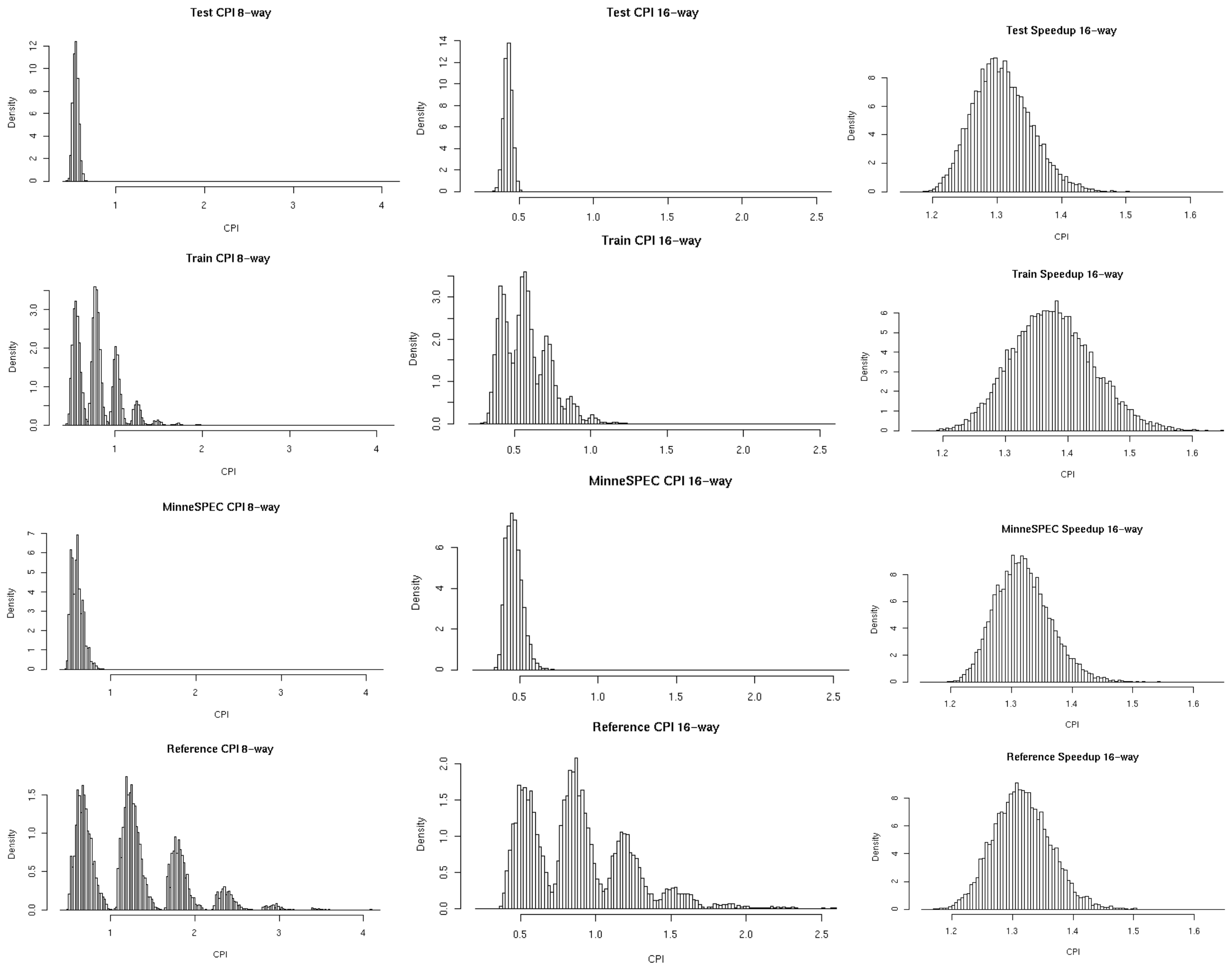


Figure 6. Bootstrapped distribution of sample mean of CPI

difference will affect the effectiveness of profile based optimization. They found that the test data set is far from the reference data set. Although the train data set is better than the test data set, it still differs from the reference data set significantly. Haskins et. al. [7] studied the difference in IPC, L1 data cache miss rate, and branch misprediction rate between train, MinneSPEC and reference data set. They concluded that the reduced input simulation can produce significant errors in important program characteristics. Eeckhout et. al [5] did similar comparison with principal component analysis and clustering analysis. They concluded that for some benchmark programs the reduced data set is representative of the reference data set whereas for others the behavior of reduced data set is quite different. Recognizing the importance of speedup, we compare the reduced data sets with reference data set in terms of both CPI and speedup. We employed statistical theory to compare the population each data set represents instead of comparing program by program. While our study confirms that the CPI is quite different between the reference and reduced data sets, we show that using reduced data set to evaluate speedup will not result in statistically different conclusion.

Viewing the small set of benchmark programs as only a sample, and calculating the confidence interval to quantify the result of performance have been demonstrated in textbooks for computer architects [9, 13]. However, this technique has seldom been used when researchers report their results based on a benchmark suite. We show the confidence interval using bootstrapping method without the normal distribution assumption. The confidence interval can guide researchers to set target accuracy when designing new simulation techniques.

7. Conclusion

In this paper, we employ statistical theory to study several topics in microprocessor simulation. We compute the autocorrelation within the instruction stream to prove that a small sampling unit (1,000 – 10,000 instructions) is more effective than large sampling unit at improving simulation accuracy, as long as the warming up overhead has not become the limiting factor. We show mathematically that the exhibited autocorrelation behavior favors small sampling units.

We have applied ratio estimator and extended previous sampling simulation method to calculate the speedup with quantifiable accuracy. Our result shows that to achieve a specified accuracy, it is not necessary to measure CPI at the same accuracy. Speedup can be accurately measured with fewer instructions sampled than CPI.

We have compared different reduced data set (test, train, and MinneSPEC). We view the SPECint suite as a random sample from the population of all CPU intensive integer benchmarks it represents. We tested the population mean of each reduced data set against the reference data set and plotted the distribution of sample mean by bootstrapping. We found that none of the reduced data sets can represent the reference data set in terms of CPI because they show different median values and widely different distributions. However, in our experiment, reduced data sets are not statistically different from the reference data set when used to evaluate the speedup. In addition, the sample mean of the speedup approximately follows the normal distribution. Confidence interval is useful for the users to evaluate the performance of computers, and for researchers to set target accuracy when designing new simulation methods.

Ideally, only a tiny portion of the full dynamic instruction stream is needed to get accurate speedup estimation, and the sampling error can be easily reduced by increasing the sample size. In reality, however, the warm-up overhead is dominant in simulation time. The error in the measurement of each sampling unit due to imperfect warm-up quickly becomes the limiting factor on accuracy. Future research in sampling simulation methodology needs to focus on more efficient and accurate warm-up mechanisms.

8. References

[1] Banks, J., Carson, J.S., and Nelson, B.L. Descrete-Event System Simulation. 2nd ed. Prentice Hall, 1999.

[2] Burger, D. and Austin, T.M. The SimpleScalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin-Madson, June 1997.

[3] Cochran, W.G. Sampling Techniques, 3rd ed. John Wiley & Sons, 1977.

[4] Conte, T. M., Hirsch, M. A. and Menezes, K. N. Reducing state loss for effective trace sampling of superscalar processors. In Proceedings of the 1996 International Conference on Computer Design (ICCD) (October 1996), 468-477.

[5] Eeckhout, L., Vandierendonck, H. and Bosschere, K.D. Quantifying the impact of input data sets on program behavior and its applications. Journal of Instruction-Level Parallelism, Volume 5, April 2003

[6] Efron, B. and Tibshirani, R.J. An Introduction to The Bootstrap. Chapman & Hall. New York 1993.

[7] Haskins, J. W. Jr., KleinOsowski, A. J., Skadron, K. and Lilja, D. J. Techniques for accurate, accelerated processor simulation: analysis of reduced inputs and sampling." Tech Report CS-2002-01, University of Virginia Dept. of Computer Science, Jan. 2002.

[8] Hsu, W.C., Chen, H., and Yew, P.C. On the predictability of program behavior using different input data sets, In Proceedings of the 6th Workshop on Interaction between Compilers and Computer Architectures, (February 2002), 45-53.

[9] Jain, R.. The Art of Computer Systems Performance Analysis. John Wiley & Sons, Inc. 1991.

[10] Jimeno-Ochoa, L.M., Ibez, P. and Vials, V. Warm time sampling: fast and accurate simulation of cache memory. In Proceedings of the 22nd. Euromicro International Conference (September 1996), 39-44.

[11] KleinOsowski, A.J. and Lilja, D.J. MinneSPEC: A new SPEC benchmark workload for simulation-based computer architecture research, Computer Architecture Letters, Volume 1, June, 2002.

[12] Lafage, T. and Seznec, A. Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream. In Proceedings of the Third IEEE Annual Workshop on Workload Characterization (September 2000), 102-110.

[13] Lilja, D.J. Measuring Computer Performance: A Practitioner's Guide. Cambridge University Press, New York, NY, 2000

[14] Perelman, E., Hamerly, G. and Calder, B. Picking statistically valid and early simulation points. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (September 2003), 244-255.

[15] Sherwood, T., Perelman, E., and Calder, B. Basic block distribution analysis to find periodic behavior and simulation points in applications. In Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (September 2001), 3-14.

[16] Sherwood T., Perelman E., Hamerly G., and Calder B. Automatically characterizing large scale program behavior. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (October 2002), 45-57.

[17] Wunderlich, R.E., Wenisch, T.F., Falsafi, B., and Hoe, J.C. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling. In Proceedings of the 30th Annual International Symposium on Computer Architecture (June 2003), 84-95.