Simulating Java Commercial Throughput Workload: A Case Study

Yue Luo and Lizy K. John
Department of Electrical and Computer Engineering, University of Texas at Austin
luo@ece.utexas.edu, ljohn@ece.utexas.edu

Abstract

Commercial throughput workloads are very important and significantly different from SPECcpu benchmarks. We present our study on the simulation methodology for one such workload, SPECjbb2000. Our result shows that the CPI correlates well with transaction throughput, thus can be used as a valid performance metric in simulation. We investigate the applicability of SimPoint technique to SPECjbb2000. It is shown that cluster analysis of BBV can successfully identify phases. With only a small number of clusters we can reap most of the benefits of such analysis. It is observed that a stationary main phase dominates the execution of the benchmark. Employing the Standardized Time Series technique in discrete event simulation theory, we propose a method to accurately measure the CPI for the main phase with only one checkpoint. The error in the result can be quantified with a confidence interval. Moreover, the simulation automatically stops when the user's accuracy requirement is met.

1. Introduction

Simulation is the most important tool for computer architects to evaluate design trade-offs. However, detailed simulation of modern benchmarks is extremely time-consuming. Various methods have been proposed to reduce the simulation time. Most of the research has been focusing on SPECcpu benchmark suite, which consists of CPU-intensive programs with a single thread. Commercial workloads such as database systems are very important in the business world, but their simulation methodology has not been studied as thoroughly. Single threaded commercial workloads largely resemble SPECcpu benchmarks. In this paper, we focus on commercial workloads that have multiple threads (or processes) and whose performance metric is throughput. We refer to them as *commercial throughput workloads*. Many important workloads fall into this category, such as web servers and OLTP applications.

These commercial throughput workloads are significantly different from SPECcpu benchmarks. The performance metric for SPECcpu is based on the entire execution time of the program. The performance of commercial throughput workloads, however, is measured in the throughput of some high-level transactions (e.g. number of database transactions per minutes) during the steady state.

The most important difference between SPECcpu and commercial throughput workloads is deterministic vs. nondeterministic behavior. SPECcpu programs are deterministic. Given the same input data set, they generate (almost) the same instruction stream in every run. Commercial throughput workloads, on the other hand, are nondeterministic. The instruction stream is different in each run on a real machine. The nondeterminism has two sources. One is the sensitivity of multithreaded workloads to the variability in the environment. In each run of the benchmark the interrupts of timer may come at slightly different time; the disk accesses may exhibit different latency. All these can affect the operating system's scheduling of the threads and change the execution path of the code. The other source of nondeterminism in commercial throughput workloads is inherent by design. These workloads are usually driven by a sequence of random numbers and the seed of the random number generator is different in each run. For instance, the sequence of requests in SPECjbb2000 is generated by a random number generator, which is seeded with the current system time. Therefore, the benchmark is designed to execute differently in each run and the performance metric of steady-state throughput only makes statistical sense in a long run.

Because of the uniqueness of the commercial throughput workloads, simulating them poses additional problems that do not exist in the simulation of SPECcpu programs. First, for SPECcpu benchmarks CPI or IPC is a valid performance metric in simulation because the instruction stream exhibits little variation from run to run. However, the relationship between

throughput of transactions and CPI may be more complicated in commercial throughput workloads, which raises the question about the validity of using CPI as the performance metric in simulation[1].

Second, the SimPoint methodology [3] is popularly used to reduce the simulation time for SPECcpu (see next section), but its applicability to commercial throughput workloads needs further investigation. Patil et. al. [10] pointed out the difficulty in applying SimPoint methodology to commercial throughput benchmarks in actual execution environment due to the nondeterminism. Another shortcoming of SimPoint is that it does not give a confidence interval for the final result to quantify the error. We want to investigate the applicability of SimPoint and (partially) solve the problem of lack of confidence interval.

The third problem is the high storage requirement in simulation. In SimPoint, predefined chunks of instructions in each phase, called *simulation points*, are simulated. There are 3 ways to simulate these predefined simulation points. In one way, the simulator can switch between fast forwarding and detailed simulation modes. It only enters detailed simulation mode while simulating the simulation points and fast-forwards all other instructions in between. The second way is to store the trace for each simulation point. And the third method is to store the state of the whole system at the start of each simulation point in a checkpoint file. The simulator can then start from the checkpoint and do an execution-driven simulation for each simulation point. Because modification in microarchitecture configuration changes the execution path, the first two methods are less accurate and checkpointing gives the most valid execution path. However, checkpointing requires large storage. In our experiment, even with compressed incremental checkpointing, each checkpoint requires 10 – 256MB space. Taking tens of checkpoints can easily amount to Gigabytes of disk storage.

The nondeterminism in commercial throughput benchmarks raises challenging problems. On the other hand, it also provides opportunity for designing better simulation methodology because the nondeterministic behavior makes the simulation amenable to the theory in discrete-event simulation. Research in discrete-event simulation studies stochastic simulation models, in which the simulation is driven by random inputs and generates output sequence that is also a stochastic process. The execution of commercial throughput workloads is driven by a sequence of random numbers and is affected by randomness in the environment. Therefore, they can and should be studied as stochastic processes. Taking advantage of the large body of discrete-event simulation theory, we can design better experiments and make rigorous statistical inference.

In this research we use SPECjbb2000 as a case study for commercial throughput workloads. SPECjbb2000 (Java Business Benchmark) evaluates the performance of server-side Java. It emulates a three-tier client/server system with emphasis on the middle tier. The multithreaded benchmark models a wholesale company with warehouses serving a number of districts. The performance metric is based on the average number of transactions processed per second.

In this paper, we make the following contributions:

- We study the validity of using CPI as the performance metric for commercial throughput workloads. We investigate the applicability of SimPoint methodology to commercial throughput workloads for identifying phases and for selecting simulation points.
- We propose a dynamic stopping rule technique based on the output analysis method in discrete-event simulation, which can quantify the CPI of the main phase (over 90% of the total execution) with a confidence interval. Moreover, the simulation can automatically stop once the user's specified accuracy requirement is met. The technique reduces storage cost by requiring only one checkpoint for the main phase.

2. Related work

Sampling is the most widely used method to reduce simulation time while retaining good accuracy. A plethora of sampling techniques has been designed. In this paper, we are interested in a group of sampling techniques, which we call *representative sampling*. In representative sampling, the entire instruction stream is divided into chunks of instructions. Some feature is extracted from each chunk, and then cluster analysis based on the feature is performed to group the chunks into clusters so that the chunks in each cluster are similar to each other. One chunk from each cluster, called a simulation point, is selected to represent the whole benchmark. SimPoint [3] is the most acknowledged representative sampling technique. In SimPoint the Basic Block Vector (BBV) of each chunk of instructions is used as the feature for cluster analysis. A basic block is a sequence of instructions in a program with a single entry point, single exit point, and no internal branches. The BBV is usually high dimensional (thousands to hundreds of thousands), and hence random projection is performed on the data to

[1] Lepak et. al. published the paper "Redeeming IPC as a performance metric for multithreaded programs" [8]. Despite the title, they focused on handling the nondeterminism in the workload and did not correlate throughput metric to IPC.

reduce the dimensionality before cluster analysis. Variance SimPoint and Early Start SimPoint [2] are two improvements to the original SimPoint. Other representative sampling techniques include SPEClite [4], RDI [1] and χ^2-distance based approach [14].

All of the above research focuses on SPECcpu benchmark suite. Patil et. al. [10] applied SimPoint methodology to commercial workloads running on Intel Itanium machines. They instrumented the code with Pin tool to gather the BBV profile of the benchmark and select the simulation points. Most of their benchmarks were run in single-threaded mode. Their result showed that representative sampling works well for such benchmarks. However, they observed that it was difficult to apply the same methodology to multithreaded commercial benchmarks in their experiment. They executed the benchmarks on real machines. Due to the nondeterminism in the multithreaded programs, the simulation points identified in one run may not exist in another run.

Alameldeen and Wood [9] studied the nondeterminism in multi-threaded workloads. They showed that because of the nondeterminism a short simulation could lead to wrong conclusion in evaluating microarchitecture changes. To deal with the nondeterminism caused by the variability in the environment, they proposed adding small artificial perturbations in memory latency.

Lepak et. al. [8] also studied nondeterminism in multithreaded workloads on multi-processor machines. They proposed a method to record synchronization events into a trace file on an execution-driven full-system simulator. An artifactual determinism-delay is injected to introduce the same synchronization event into a new simulation.

3. Experiment setup

We use Simics [13], a commercial full system simulator, to simulate a SunFire server running Solaris 9. The clock frequency of the processor is set to 1GHz. Using the microarchitecture model in Simics, we modeled two configurations with different cache latencies as shown in Table 1.

We run SPECjbb benchmark with two different setups, 2 warehouses and 30 warehouses. In this way, we can see how our simulation methodology performs with a wide range of benchmark parameters. For the small setup, the heap size is fixed at 256MB, and for the big setup, the heap size is initially 768MB and allowed to increase to a maximum of 1.5GB.

We only study the behavior when the benchmark is in steady state. This can be done by doing cache simulation in functional mode until the ramp-up period of SPECjbb has ended. Then we take a checkpoint and start Micro Architectural Interface (MAI) mode simulation. Before taking performance measurement, we also warm up the microarchitecture structures (caches, branch predictor, etc) [5][6][7].

Table 1. System configurations

		Configuration 1	**Configuration 2**
Processor		Clock frequency 1GHz Fetches, executes, retires, and commits 4 instructions per cycle	
L1 instruction cache	Organization	32 Kbytes, 8-way set associative, 64 bytes/line	
	Latency	1 cycle	2 cycles
L1 data cache	Organization	32 Kbytes, 8-way set associative, 64 bytes/line	
	Latency	1 cycle	2 cycles
L2 cache	Organization	2 Mbytes, 8-way set associative, 64 bytes/line	
	Latency	6 cycles	9 cycles
Memory		2 Gbytes, 85 cycle latency	2 Gbytes, 99 cycle latency

4. Phase identification

In this section we examine the overall behavior of SPECjbb and try to answer the questions: 1. Is CPI still a valid performance metric? 2. Is SimPoint methodology still applicable to simulating SPECjbb?

We first correlate CPI to the high level throughput. The throughput we use is number of transactions per million cycles. We measure the throughput and CPI for the two configurations. We then calculate the speedup in throughput and in CPI and compute the difference in the two speedups. If the difference is zero, then CPI will be a performance metric equivalent to throughput. The result is shown in Table 2. The difference is small, about 1.68% for 30-warehouse setup and 1.06% for 2-warehouse setup. Therefore, we would conclude that CPI and throughput correlate well in the simulation of SPECjbb, the difference between speedup in throughput and speedup in CPI is very small and can probably be ignored in most practical uses. In the rest of our paper, we use CPI as our main performance metric. One caveat here is that SPECjbb has almost no I/O activity. If a benchmark exhibits significant I/O activity such as disk accesses, then a faster processor may spend (relatively) more time waiting for I/O, and CPI may be a poor indicator for throughput.

We then get an overview of the CPI graph of the benchmark. We measure the CPI for every chunk of 1 million instructions. Figure 1a and 1b show respectively the CPI graph for 30-warehouse setup and for 2-warehouse setup running on configuration 1. Our simulation is too long to display clearly so only part of

the result is shown. The results running on configuration 2 are very similar thus not shown here.

Table 2. Speedup for throughput and CPI for configure 1 vs. configuration 2.

Warehouses	30		2	
Configuration	2	1	2	1
Transactions	112572	112429	143560	135104
Cycles (10^6)	33374	20474	36893	215305
Throughput (trans/10^6cycles)	3.3730	5.4913	3.8912	6.27490
Speedup in throughput	1.628		1.613	
CPI	3.3031	2.6352	3.0804	1.9307
Speedup in CPI	1.601		1.595	
Difference between speedups	1.68%		1.06%	

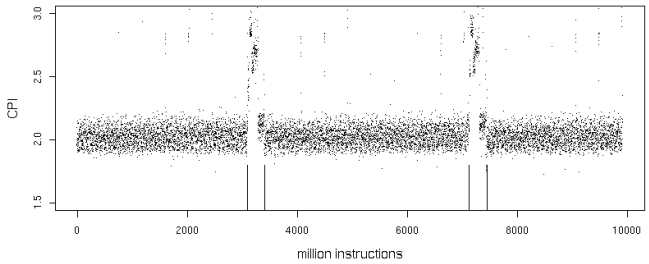

Figure 1a. CPI of every 1 million instructions for 30-warehouse setup on configuration 1.

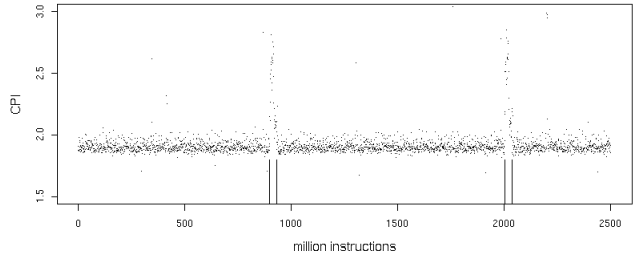

Figure 1b. CPI of every 1 million instructions for 2-warehouse setup on configuration 1.

On the first order, the execution can be viewed as a stationary phase interrupted periodically by some unusual activity with very high CPI. We call that stationary phase the *main phase*. We suspect that the high CPI activity is the garbage collection, so we develop a profiler through Java Virtual Machine Profiler Interface (JVMPI). Upon the start and stop of the garbage collection, the profiler generates marks in our trace file. The short vertical lines in Figure 1a and 1b mark the boundary of the garbage collection activity. As we can see, the high CPI activity is aligned with the garbage collection. Further instrumentation of the benchmark shows that during the garbage collection period, no transactions are completed. Therefore, the execution of SPECjbb consists of the main phase of transaction processing and periodical garbage collection.

We then examine the effectiveness of applying a methodology similar to SimPoint. The BBV for each 1 million-instruction chunk is collected. K-means cluster analysis is performed to group them into clusters of similar chunks. Each cluster corresponds to a phase. For SimPoint technique to work well, the CPI values of the chunks in the same cluster/phase should be very close to each other. In other words, the CPI values in each cluster/phase should be homogeneous. To evaluate the homogeneity of the clustering result, we use Normalized Standard Deviation (NSD), defined as

$$NSD = \sqrt{(\sum_{i=1}^{k} \frac{n_i}{n} S_i^2)} / S \,,$$

where n_i is the size of cluster i. S_i is the standard deviation of CPI in cluster i. n is the total number of chunks and S is the standard deviation of CPI for all chunks. The lower NSD, the more homogeneous the phases are. NSD is a statistically sound metric. It shows how much we have gained from taking the extra effort to do the clustering compared to simple random sampling[2]. If NSD is close 1, then the CPI in each cluster is as varied as in the whole benchmark. It is like randomly grouping data points into clusters and no reduction in variance is achieved in each phase. On the other hand, if NSD is close to 0, then the CPI remains almost constant in each phase and our result will be much more accurate than simple random sampling.

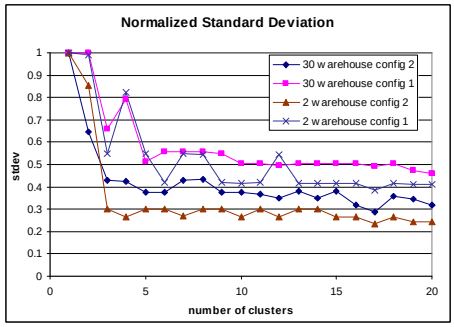

Figure 2. Normalized standard deviation for CPI for different number of clusters.

We vary the number of clusters from 2 to 20. The NSD result is shown in Figure 2. Many of the values are much smaller than 1. Therefore, doing clustering can significantly improve the accuracy of the sampling result. We also observe that NSD drops quickly at the beginning of the curve and then decreases slowly. This means that by dividing the benchmark into a small number (3-5) of clusters/phases, we can capture most of the benefit of doing clustering analysis. This is

[2] More formally, NSD^2 is the ratio of the variation in the sample mean for proportional stratified sampling to the variation in the sample mean of simple random sampling with the same sample size. See [17] for detail.

consistent with the CPI graphs in Figure 1, which does not exhibit a large number of phases visually.

The clustering is done with microarchitecture independent BBVs whereas the phases we see in Figure 1 are for CPI, which is microarchitecture dependent. It has been demonstrated that for SPECcpu programs clusters correspond well with CPI phases [3]. We would like to verify it for SPECjbb. With a small number of clusters, this can be inspected visually. Figure 3a shows the CPI for each chunk of instructions and Figure 3b shows the cluster number each chunk belongs to. To make the figure legible, only a small part of the total clustering result is shown. Figure 4 shows the same result for 30-warehouse setup on configuration 1. The cluster analysis generally identified the main phase and the garbage collection phase correctly. Compared with marking the garbage collection activity with JVMPI, the cluster analysis has certain advantages. First, it can further identify the phases within garbage collection. It is clear in Figure 4a that there are phases within garbage collection, and the cluster analysis identified them as shown in Figure 4b. Second, cluster analysis can identify sporadic outliers in the main phase as we can see in both Figure 3 and Figure 4. However, as shown in Figure 4, clustering result is not 100% correct. It mistakes part of the garbage collection as the main phase.

One problem with SimPoint is that it does not give a confidence interval for the final result. Thus it is hard for the user to quantify the error. One way to solve this problem is to use stratified sampling [17][18]. However, stratified sampling requires multiple data points from each cluster to generate an accurate estimation of confidence interval, resulting in large storage cost for checkpoints.

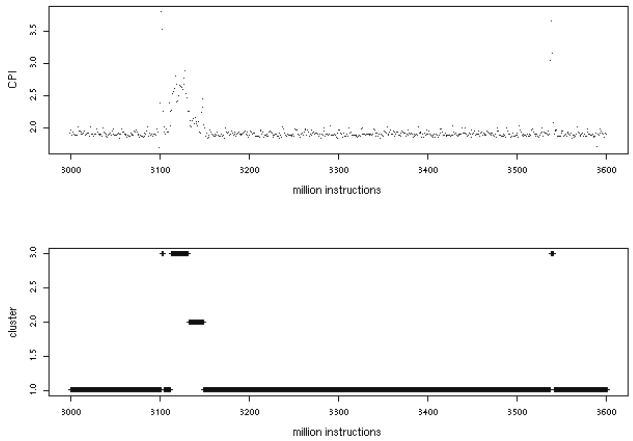

Figure 3 CPI and cluster number for each chunk of instructions for 2-warehouse setup on configuration 1.

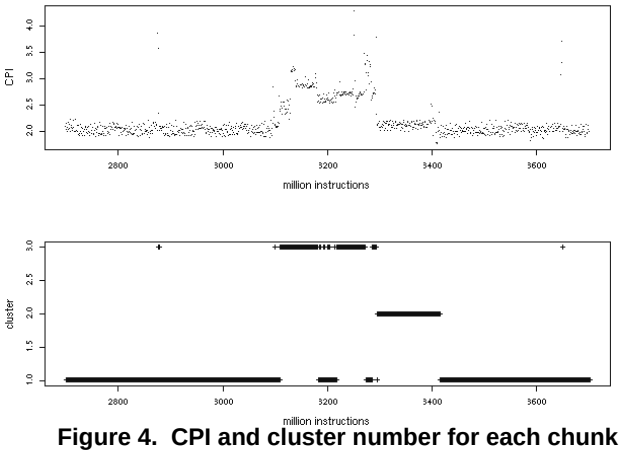

Figure 4. CPI and cluster number for each chunk of instructions for 30-warehouse setup on configuration 1.

In the next section, we propose a solution based on output analysis techniques in discrete event simulation theory. Our goal is to 1) put a confidence interval on the simulation result and make the simulation automatically stop based on the user's accuracy requirement, and 2) store as few checkpoints as possible.

5. Dynamic stopping rule for simulating throughput benchmarks

In discrete-event driven simulation theory, the commercial throughput workloads belong to nonterminating simulation, where there is no natural event to specify the length of the simulation and where the user only cares about the steady state performance. Nonterminating processes are usually stationary.

Looking at Figure 1a and 1b, we can see that there is no steady state distribution in CPI because of the periodical garbage collection activity. However, if we ignore the garbage collection and focus only on the main phase, it is a stationary process. In the main phase the program processes the transaction requests, which are generated by a sequence of random numbers. The sequence of random numbers is stationary and thus the characteristic of the output CPI process driven by the random numbers is also in a state of statistical equilibrium.

To identify the main phase, we first use the cluster analysis. We group the BBVs into 3 clusters, from which we can identify the main cluster. However, as shown in Figure 4, some garbage collection activity can be misclassified into the main cluster, so we also use JVMPI to mark the beginning and the end of garbage collection. In this way we get the main phase after removing the garbage collection and some outliers. Table 3 shows the percentage of the main

phase in the whole benchmark execution. In all cases, more than 90% of the instructions executed are in the main phase. Therefore, from now on, we focus only on the main phase and try to design a simulation methodology for it.

Table 3. Percentage of instructions executed in the main phase

Warehouses	Configuration	Percentage in main phase
2	1	95.6%
2	2	95.3%
30	1	92.6%
30	2	92.0%

Constructing the confidence interval for the mean performance metric is a tricky problem because the CPI values have autocorrelation. There are usually three ways to do it. In the first method, the benchmark is run once for such a long period of time that in any practical use the mean CPI from the long run is deemed the true mean. Because the user cannot really afford such a long run, sampling is done within the run. Confidence interval can be constructed from the sampling result. The second method is to do multiple relative short runs of the benchmark. In each run a different sequence of random numbers is used. Not only the random number driving the artificial variability in memory latency should be changed, the random seed in the benchmark that generates the sequence of transaction requests must also be varied. Because the multiple runs are statistically independent, the confidence interval can be constructed with classical statistics for independent data. These two methods suffer from the problem of requiring many checkpoints. The third method is to try constructing a confidence interval from just one simulation of moderate length. Unlike the previous two methods, only one checkpoint is needed. The length of a single simulation run is sequentially increased until an "acceptable" confidence interval can be constructed, thus it is often called *sequential procedure*. There are several techniques for deciding when to stop the simulation run.

Different techniques make different assumptions about the autocorrelation, so a technique usually works well only in situations where its assumptions are valid. We use the dynamic stopping rule based on Standardized Time Series (STS) proposed by Schruben [11]. The assumption of STS is weak. The main assumption is phi-mixing. An intuitive explanation is that a phi-mixing process is random enough so that the distant future is essentially independent of the past or present. This is consistent with our observation of temporal locality. With temporal locality, the behavior of the program (e.g. address of memory accesses) in the near future will be similar to the present. As the distance increase, the temporal locality becomes weaker. And two chunks of instructions far away should be independent. In fact, Schruben demonstrated that almost all stationary finite state discrete-event computer simulations satisfy the assumptions for STS [11].

In STS all data points are divided into b batches each containing m consecutive data points, so that the total number of data points are $n=b*m$. The confidence interval can be calculated as follows.

$$CI = \bar{\bar{Y}} \pm t_{1-\alpha/2,df} \frac{\hat{\sigma}}{\sqrt{n}}$$

where

$$\bar{\bar{Y}} = \frac{1}{n}\sum_{i=1}^{n} Y_i = \frac{1}{b}\sum_{i=1}^{b} \bar{Y}_{i,m}$$

$\bar{Y}_{i,m}$ = average of the ith batch of size m; t=value of the Student t distribution; α=significance level; and df=degrees of freedom=$2b$-1;

$\hat{\sigma}$ =estimate of σ, given by $\hat{\sigma}$ =$A/(2b-1)$

where $A = \sum_{i=1}^{b} [(12A_i^2)/(m^3 - m) + m(\bar{\bar{Y}} - \bar{Y}_{i,m})^2]$

A_i can be computed by the recursion

$$A_i(k+1) = A_i(k) + 1/2(kY_{i,k+1} - \sum_{j=1}^{k} Y_{i,j}) \text{ , } k=1, .. , m$$

with $Y_{i,j}$=jth observation of the ith batch, and $A_i(0)=0$

Based on the above discussion, we propose the following procedure to measure the CPI for the main phase of commercial throughput workloads.

1. Use cluster analysis of BBVs and profiling to identify the main phase. Find a long consecutive part of the phase and dump a checkpoint at the beginning.
2. Starting from the checkpoint, warm up the microarchitecture and simulate m_0*b chunks of instructions.
3. Calculate the confidence interval. If it meets the user's accuracy requirement, stop.
4. Continue simulating additional $\Delta m*b$ chunks of instructions (i.e. $m=m+\Delta m$), go to step 3.

We need a long consecutive part of the main phase in step 1 because the changes in microarchitecture can affect the execution path. A long consecutive part makes it very unlikely that our simulation will stray to a different phase on a different microarchitecture configuration. We start with batch size m_0. It is not advisable to set m_0=1 because the confidence interval from STS is not very accurate when the simulation is too short. Δm is a tradeoff between calculation and simulation. If Δm is very small, the simulation can stop as soon as the result meets the user's accuracy requirement, but the calculation of confidence interval may be done many times along the way. On the other hand, if Δm is large, we may do more simulations than

necessary but we don't need to do as many calculations of confidence interval.

This method has several advantages. First, it gives the confidence interval to quantify the simulation result of the main phase. Second, the simulation can stop when it meets the user's accuracy requirement. Third, it needs only one checkpoint for the entire main phase.

We conduct an experiment to evaluate the dynamic stopping rule in the simulation of SPECjbb. The chunk size is reduced to 1000 instructions because STS requires many chunks to give an accurate result. We set *b*=15 following [12]. Figure 5 shows the relative error limit (i.e. half confidence interval divided by the mean) at 95% confidence level for different simulation lengths. The confidence interval generally becomes narrower as the simulation length increases. Different setups show different decreasing rate but within 10 million instructions, all the errors reach or go below 2%. As an example, suppose that the user chooses an accuracy requirement of 3% relative error with 95% confidence. Table 3 shows the simulation length required as well as the real error. (When calculating the real error we assume that the mean CPI of the main phase of the whole simulation is the true mean.) By simulating a few million instructions, the relative error limit can be reduced to within 3%, and the real error is indeed within the specified limit.

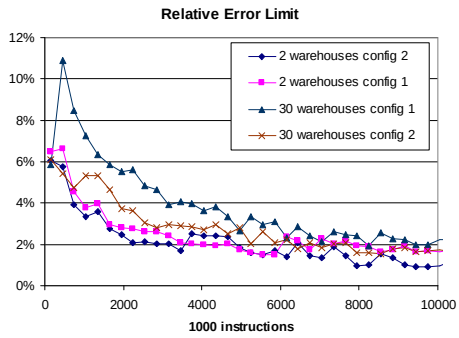

Figure 5. Relative error limit for the main phase at 95% confidence level for different simulation length.

The above result shows the accuracy of our method for the main phase, which is the major part of the benchmark execution. Next, we evaluate the overall accuracy. To get the overall mean CPI, we use cluster analysis of BBV on the remaining part of the benchmark execution. Analysis similar to previous section shows that about 3 clusters should reap most of the benefit of clustering. Therefore, a total of 4 checkpoints are needed, 1 for the main phase, and 3 for the remaining part. Checkpoints for the main phase are simulated for the length listed in Table 4. The other checkpoints are simulated for 1 million instructions each. Table 5 gives the result for the overall accuracy. For comparison, the result based on SimPoint 2.0 is also shown. Even with simulating fewer instructions our relative error is much smaller. The errors are smaller than those in Table 4, which indicates that the error in main phase and error in the remaining part happened to cancel each other in the experiment. To remove this "good luck", in column 4 we show the maximum error if the two errors would add to each other. This maximum error is still smaller than SimPoint. The benefit comes from the accurate estimation of the CPI for the main phase, which comprises over 90% of the total instructions.

Table 4. Simulation length to achieve 3% relative error limit at 95% confidence level, and the real error in CPI.

Warehouses	Configu-ration	Simulation length (10^6 instructions)	Real error
2	1	1.65	1.2%
2	2	1.65	0.7%
30	1	5.55	1.4%
30	2	2.85	1.3%

Table 5. Comparing the overall simulation length and accuracy with SimPoint

Ware-houses	Config-uration	Dynamic stopping rule			SimPoint	
		Simulation length (10^6 instr)	Max relative error	True relative error	Simulation length (10^6 instr)	True relative error
2	1	4.65	2.4%	0.82%	13	3.9%
2	2	4.65	3.2%	0.20%	20	5.1%
30	1	8.55	2.9%	0.85%	10	3.2%
30	2	5.85	4.0%	0.31%	15	5.5%

6. Conclusions and future work

Commercial throughput workloads are very important in the business world. They differ significantly from SPECcpu benchmarks. Because of multithreading, the execution of the benchmark is nondeterministic. However, the simulation methodology for them has not been well studied as for SPECcpu benchmarks.

We have studied one commercial throughput workload, SPECjbb2000. It is observed that CPI correlates well with transaction throughput. Therefore, CPI is a valid performance metric in the simulation. We applied SimPoint methodology to SPECjbb2000. BBV can be used successfully to identify the phases of the benchmark. Our result shows that a small number of phases can capture most of the benefit. The main phase, which consists of transaction processing, dominates the execution. Therefore, to get accurate final results, accuracy for the CPI in main phase is most important.

We propose a method based on the Standardized Time Series technique to simulate the main phase. The

chunk size should be set such that the main phase is stationary. The stationary main phase is identified using a combination of code profiling and cluster analysis of BBV. With only one checkpoint, the user can get very accurate estimation of the CPI for the main phase. The simulation can give a confidence interval to quantify the error in the result. Moreover, the simulation automatically stops when it has met the user's accuracy requirement.

This paper presents our pilot study on the simulation methodology for commercial throughput workloads. SPECjbb2000 is simple compared to other server workloads. It does not have disk accesses or network I/O activity. We are applying our proposed technique on more complex workloads such TPC-C. Furthermore, in most situations, the user cares more about the performance improvement (i.e. speedup) resulting from the microarchitecture change than the absolute value of CPI. It has been shown that measuring speedup may require shorter simulation time than measuring CPI [15][16]. We plan to extend our method to measure speedup with confidence interval and to further reduce the simulation time.

References

[1] T. Lafage, and A. Seznec. 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.

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

[3] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. 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.

[4] R. Todi. SPEClite: Using Representative Samples to Reduce SPEC CPU2000 Workload. IEEE 4th Annual Workshop on Workload Characterization. 2001.

[5] J. W. Haskins, Jr. and K. Skadron. "Memory Reference Reuse Latency: Accelerated Warmup for Sampled Microarchitecture Simulation." In Proceedings of the International Symposium on Performance Analysis of Systems and Software, Mar. 2003.

[6] L. Eeckhout, S. Eyerman, B. Callens, and K. De Bosschere. Accurately warmed-up trace samples for the evaluation of cache memories. In Proceedings of the 2003 High Performance Computing Symposium (HPC-2003), pages 267–274, Apr. 2003.

[7] Y. Luo, L. K. John, and L. Eeckhout. Self-Monitored Adaptive Warm Up. In Proceedings of 16th Symposium on Computer Architecture and High Performance Computing. Oct. 2004.

[8] K. Lepak, H. W. Cain, and M. H. Lipasti. Redeeming IPC as a Performance Metric for Multithreaded Programs. Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques. 2003.

[9] A. R. Alameldeen and D. A. Wood. Variability in Architectural Simulations of Multi-threaded Workloads. Proceedings of the Ninth International Symposium on High-Performance Computer Architecture. 2003.

[10] H. Patil, R. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing Representative Portions of Large Intel Itanium Programs with Dynamic Instrumentation. Proceedings of the 37th International Symposium on Microarchitecture. 2004.

[11] L. Schruben. Confidence Interval Estimation Using Standardized Time Series. Operations Research. Vol 31, No. 6, pp 1090-1107. 1983.

[12] R. R. Duersch and L. W. Schruben. An Interactive Run Length Control for Simulations on PCs. Proceedings of the 1986 Winter Simulation Conference, pp 866-870. 1986.

[13] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, B. Werner, Simics: A Full System Simulation Platform, Computer, February 2002.

[14] R. Srinivasan, J. Cook, S. Cooper. Fast, Accurate Microarchitecture Simulation Using Statistical Phase Detection. 2005 IEEE International Symposium on Performance Analysis of Systems and Software. 2005

[15] M. Ekman, P. Stenstrom. Enhancing Multiprocessor Architecture Simulation Speed Using Matched-Pair Comparison. 2005 IEEE International Symposium on Performance Analysis of Systems and Software. 2005.

[16] Y. Luo and L. K. John. Efficiently Evaluating Speedup Using Sampled Processor Simulation. Computer Architecture Letters, Volume 3, Sept. 2004.

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

[18] R. E. Wunderlich, T. F. Wenisch, B. Falsafi and J.s C. Hoe. An Evaluation of Stratified Sampling of Microarchitecture Simulations. IEEE Workshop on Duplicating, Deconstructing, and Debunking, June 2004.