Speed versus Accuracy Trade-Offs in Microarchitectural Simulations

Joshua J. Yi, *Member*, *IEEE*, Resit Sendag, *Member*, *IEEE*,
David J. Lilja, *Fellow*, *IEEE*, and Douglas M. Hawkins

Abstract—Due to the long simulation time of the `reference` input set, computer architects often use reduced time simulation techniques to shorten the simulation time. However, what has not yet been thoroughly evaluated is the accuracy of these techniques relative to the `reference` input set and with respect to each other. To rectify this deficiency, this paper uses three methods to characterize reduced input set, truncated execution, and sampling-based simulation techniques while also examining their speed versus accuracy trade-off and configuration dependence. Our results show that the three sampling-based techniques, SimPoint, SMARTS, and random sampling, have the best accuracy, the best speed versus accuracy trade-off, and the least configuration dependence. On the other hand, the reduced input set and truncated execution simulation techniques had generally poor accuracy, were not significantly faster than the sampling-based techniques, and were severely configuration dependent. The final contribution of this paper is a decision tree, which can help architects choose the most appropriate technique for their simulations.

Index Terms—Modeling of computer architecture, measurement techniques, modeling techniques.

1 Introduction

The SPEC CPU2000 benchmark suite [12] is a commonly used suite for simulation-based computer architecture research and the largest input set for each benchmark is the `reference` input set. Although this input set typically yields the most realistic behavior, due to its very long simulation time, architects rarely simulate it to completion.

Since lengthy simulation times preclude a detailed exploration of the design space, architects resort to alternative simulation techniques to reduce the simulation time. These techniques include reducing the size of the input set, simulating a single piece of the program that ostensibly represents the whole program, and sampling. Although these techniques reduce the simulation time, what is not clear is how the characteristics and the accuracy of each technique compare to the `reference` input set and to each other. Without thoroughly understanding the effects that these techniques can have on the results, the validity of those results is suspect, which nullifies the point of performing the simulations in the first place.

- *J.J. Yi is with the Networking and Computing Systems Group, Freescale Semiconductor, Inc., 7700 West Parmer Lane, MD: PL30, Austin, TX 78729. E-mail: jjyi@ece.umn.edu.*
- *R. Sendag is with the Department of Electrical and Computer Engineering, University of Rhode Island, 4 East Alumni Avenue, Kingston, RI 02881. E-mail: sendag@ele.uri.edu.*
- *D.J. Lilja is with the Department of Electrical and Computer Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455. E-mail: lilja@umn.edu.*
- *D.M. Hawkins is with the School of Statistics, University of Minnesota, 224 Church Street S.E., Minneapolis, MN 55455. E-mail: doug@stat.umn.edu.*

Manuscript received 19 Aug. 2006; revised 25 Mar. 2007; accepted 26 Apr. 2007; published online 6 June 2007.
Recommended for acceptance by Z. Xu.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0322-0806.
Digital Object Identifier no. 10.1109/TC.2007.70744.

To address this issue, this paper evaluates the accuracy of the seven most prevalent techniques with respect to the `reference` input set by characterizing them using three different methods. The seven techniques are

1. SimPoint [17],
2. SMARTS [18],
3. random sampling [4],
4. reduced input sets (MinneSPEC [13] and SPEC `test` and `train`),
5. simulating the first `Z` million instructions only,
6. fast-forwarding `X` million instructions and then simulating the next `Z` million, and
7. fast-forwarding `X` million, warming up the processor for the next `Y` million, and then simulating the next `Z` million instructions.

The three methods that we used to characterize these seven techniques are the 1) processor bottleneck, 2) execution profile, and 3) architectural-level characterizations. We then analyze each technique from three additional perspectives, namely, their speed versus accuracy trade-off (SvAT), the potential configuration dependence, and the fidelity of their performance bottlenecks. Finally, based on the results of these characterization methods and analyses, we present a decision tree that architects can use to determine the technique that best suits their situation.

The remainder of this paper is organized as follows: Section 2 describes the problem in addition to describing each of the seven techniques. Sections 3 and 4 describe the experimental framework and characterization methods, respectively, while Sections 5 and 6 present the results. Section 7 describes some related work, Section 8 makes specific recommendations about simulation methodology and Section 9 summarizes the paper.

2 Prevailing Simulation Techniques

To reduce the simulation time to a tractable level, computer architects typically use several techniques to approximate the behavior of the `reference` input set. These techniques fall into three categories: 1) *reduced input sets*, 2) *truncated execution*, and 3) *sampling*.

Reduced input sets. The basic idea behind reduced input sets is to modify the `reference` input set in some way to reduce the simulation time when using the modified input set. The hope is that the reduced input set still retains the characteristics of the `reference` input set, but with a lower simulation time. The primary advantage of using reduced input sets is that the entire behavior of the program is simulated in detail, including initialization, the main body of the computation, and cleanup. The main disadvantage is that their results may be very dissimilar compared to those produced by the `reference` input sets. In addition, developing reduced input sets can be a very tedious and time-consuming undertaking. Examples of the SPEC 2000 reduced input sets include the **`test`** and **`train`** input sets from SPEC and the **MinneSPEC** **`small`**, **`medium`**, and **`large`** reduced input sets [13].

Truncated execution. In truncated execution, the architect simulates the benchmark for a fixed number of instructions while presuming that that arbitrary interval is representative of the entire program. There are three primary variations. In the simplest case, which we call **Run `Z`**, only the first `Z` million instructions of the benchmark are simulated using the `reference` input set, where the value of `Z` determines the simulation time. A variation on this idea is to fast-forward through the first `X` million instructions and then switch to detailed simulation for the next `Z` million (that is, **Fast-Forward `X` + Run `Z`** (**FF `X` + Run `Z`**)). This technique potentially improves on Run `Z` by skipping over the less interesting aspects of the program. One problem with the FF `X` + Run `Z` technique is that, after fast-forwarding, the processor and memory states are "cold" (that is, invalid). The solution to this problem is to "warm up" the processor and memory before starting detailed simulation. One simple implementation is to perform detailed simulation for `Y` + `Z` million instructions after fast-forwarding, while tracking the simulation statistics for only the last `Z` million. We refer to this technique as **Fast-Forward `X` + Warm Up `Y` + Run `Z`** (**FF `X` + WU `Y` + Run `Z`**).

Sampling. Population sampling is a statistical technique that is used to infer the characteristics of the population by extrapolating from the characteristics observed in a subset. The key to good results with population sampling is to ensure that the chosen subset accurately reflects the overall population. Computer architects have proposed using *representative*, *periodic*, and *random* sampling as the basis for reduced time simulation techniques.

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. With the **SimPoint** [17] technique, for example, a relatively small number of simulation points are chosen to be the representative of the behavior of the entire program. Determining the simulation points first involves profiling the benchmark to identify the candidate simulation points and then using machine-learning-based clustering to select a set that is representative of the entire program. After simulation, the results from each simulation point are weighted to compute the final simulation results. The number of simulation points and the length of each determine the overall simulation time.

By contrast, *periodic sampling* simulates selected portions of the dynamic instruction execution at fixed intervals: **SMARTS** [18] is a recent example. The sampling frequency and the length of each sampling unit are used to control the overall simulation time. To improve its accuracy, SMARTS uses a statistical sampling theory to estimate the CPI error of the sampled simulation versus the `reference` simulation. If the estimated error is higher than the user-specified confidence interval, then SMARTS recommends a higher sampling frequency. SMARTS also uses "functional warming" to maintain the branch predictor and cache state between sampling units.

Finally, in *random sampling*, the simulation results from N randomly chosen and distributed intervals are combined together to produce the overall simulation results. To reduce the error associated with random sampling, Conte et al. [4] suggested increasing the number of instructions dedicated to processor warm-up before each sampling unit and/or increasing the number of sampling units.

Prevalence of simulation techniques. In addition to simulating the `reference` input set to completion and the above techniques, a multiplicity of additional permutations exist. For obvious reasons, quantifying the accuracy of all permutations is infeasible. Therefore, to determine the set of techniques to analyze in this paper, we examined the proceedings for HPCA, ISCA, and MICRO from 1994 to 2003 to determine the most prevalent techniques. Our results show that the four most popular techniques are FF `X` + Run `Z` (27.3 percent of all known techniques), Run `Z` (23.1 percent), *reduced input sets* (18.5 percent), and simulating the benchmark to completion (17.8 percent). Since these four techniques account for almost 90 percent of all known techniques, we included these four techniques in the set of candidate techniques studied in this paper. We also included FF `X` + WU `Y` + Run `Z` since it is a more accurate version of FF `X` + Run `Z`, SimPoint and SMARTS since they are likely to increase in popularity, and, although it was rarely used, random sampling since it makes for an interesting comparison to the other sampling-based techniques.

Table 1 shows our final list of the 81 permutations of the candidate techniques. For the truncated execution techniques, the values of `X`, `Y`, and `Z` were based on the superset of common permutations that we found in our survey, whereas the specific values for the sampling-based techniques were based on those in [11] for SimPoint, [18], [19] for SMARTS, and [4], [5], [6] for random sampling. (To facilitate the comparison of SMARTS and random sampling, the values for random sampling were set to SMARTS-like values.)

For random sampling, the number of instructions between sampling units was randomly chosen from a range of values (either 10^*U to $10,000^*$U or 10^*U to $100,000^*$U, where U is the number of instructions in each sampling unit) and where there is an equal likelihood of choosing any

TABLE 1
Configurations for the Simulation Techniques ($X + Y \bmod 100M = 0$)

Number of Permutations	Technique	Permutations
3	**SimPoint (Standard [17])**	Single 100M, multiple 10M (max_K: 100), and 100M (max_K: 10) SimPoint 1.0, 7 Random Seeds (seedproj = 1), 100 iterations Warm-Up: 1M for 10M and 0M for 100M; assume cache hit [11]
9	**SMARTS**	Detailed simulation length per sampling unit (U): 100, 1000, 10000 Warm-up length per sampling unit (W): 200, 2000, 20000 Initial number of sampling units (n): 10,000 Configuration: 99.7% confidence level, ±3% confidence interval [18]
12	**Random Sampling**	Detailed simulation length per sampling unit (U): 100, 1000, 10000 Warm-up length per sampling unit (W): 100, 10000 Range of random periods (k): 10*U to 10,000*U (10K) and 10*U to 100,000*U (100K) Results based on the average of 3 runs
3-5	**Reduced**	MinneSPEC `small`, `medium`, `large`
		SPEC `test`, `train`
4	**Run Z**	`Z:` 500M, 1000M, 1500M, 2000M
12	**FF X + Run Z**	`X:` 1000M, 2000M, 4000M
		`Z:` 100M, 500M, 1000M, 2000M
36	**FF X + WU Y + Run Z**	`X:` 999M, 1999M, 3999M; 990M, 1990M, 3990M, 900M, 1900M, 3900M
		`Y:` 1M; 10M, 100M
		`Z:` 100M, 500M, 1000M, 2000M

TABLE 2
SPEC 2000 Benchmarks and Input Sets

Benchmark	small	medium	large	test	train	reference
gzip	smred.log	mdred.log	lgred.log	test.combined	train.combined	ref.log
vpr-Place	smred.net	mdred.net	N/A	test.net	train.net	ref.net
vpr-Route	small.arch.in	small.arch.in		small.arch.in	train.arch.in	ref.arch.in
gcc	smred.c-iterate.i	mdred.rtlanal.i	N/A	cccp.i	cp-decl.i	166.i
art	N/A	N/A				-startx 110
mcf	smred.in	N/A	lgred.in	test.in	train.in	ref.in
equake	N/A	N/A	lgred.in	test.in	train.in	ref.in
perlbmk	smred.makerand	mdred.makerand	N/A	N/A	scrabbl	diffmail
vortex	smred.raw	mdred.raw	lgred.raw	test.raw	train.raw	lendian1.raw
bzip2	N/A	N/A	lgred.source	test.random	train.compressed	ref.source

value from the range. Each permutation was simulated three times and the results were averaged across all three simulations.

3 Experimental Framework

In this paper, we used `wattch` [2] as the base simulator. We chose `wattch` as the base simulator because we originally wanted to evaluate the performance and power accuracy of each simulation technique. However, due to the fact that there is less variation in the power results, the power accuracy of all techniques was comparatively higher (and, therefore, less interesting) than the performance results. Consequently, due to space limitations, we do not present those results. We modified `wattch` to include user-configurable instruction execution latencies and throughputs and a user-configurable warm-up. To implement SMARTS, we added periodic sampling, functional warming, and statistical error estimation to `wattch`.

To characterize the accuracy of each technique, we used a total of 56 different processor configurations. Since these configurations are associated with a specific characterization method, the configurations are listed in Sections 4.1 and 4.3, along with the characterization method.

The 10 benchmarks that were used in this study, shown in Table 2 along with their input sets, were selected from the SPEC 2000 benchmark suite because they are all written in C and because these benchmarks represent the most popular benchmarks that architects typically use [3]. The total simulation time limited the number of benchmarks that we could simulate. Even then, simulating the `reference` input set and the 81 permutations in Table 1 for 56 processor configurations and 10 benchmarks required the simulation of over *one quadrillion* (10^{15}) detailed instructions, which required approximately four years of constant simulation. All benchmarks were compiled at optimization level O3 by using SimpleScalar's version of the `gcc` compiler version 2.6.3. With the exception of the reduced input sets, the input set for all techniques was the `reference` input set or one of the `reference` input sets in the case of *gzip*, *gcc*, *perlbmk*, *vortex*, and *bzip2*.

4 Description of the Characterization Methods

To measure the accuracy of each technique, we used three different characterization methods. Section 4 describes these methods, while Section 5 presents the results of each.

TABLE 3
Processor Configurations Used for the Architectural-Level Characterization

Parameter	Config #1	Config #2	Config #3	Config #4
Decode, issue, commit width	4-way		8-way	
Branch predictor, BHT entries	Combined, 4K	Combined, 8K	Combined, 16K	Combined, 32K
ROB/LSQ entries	32/16	64/32	128/64	256/128
Int/FP ALUs (Mult/div units)	2/2 (1/1)	4/4 (4/4)	6/6 (4/4)	8/8 (8/8)
L1 D-cache size (KB), assoc, lat (Cycles)	32, 2-way, 1	64, 4-way, 1	128, 2-way, 1	256, 4-way, 1
L2 cache size (KB), assoc, lat (Cycles)	256, 4-way, 10	512, 8-way, 7	1024, 4-way, 15	2048, 8-way, 12
Memory lat (Cycles): First, following	150, 10	100, 5	300, 20	200, 10

4.1 Processor Bottleneck Characterization

The first characterization method is a performance bottleneck analysis using a Plackett and Burman (PB) design [16]. For architects, the PB design can determine which processor and memory parameters have the largest effect on the performance of the processor, that is, the biggest performance bottlenecks. The output of a PB design is a value that is associated with each input parameter (bottleneck). The magnitude of this number represents the effect that that parameter has on the variability in the output value, for example, number of cycles. The parameters with the largest PB magnitudes have the largest effect on the number of cycles and represent the largest performance bottlenecks in the processor and memory subsystem.

After calculating the effect that each parameter has on the CPI, we rank the parameters based on their PB magnitudes (1 = largest magnitude) and then vectorize the ranks. To determine the similarity in the performance bottlenecks of the `reference` input set and each technique, we calculate the euclidean distance between their rank vectors. Therefore, the technique that has the smallest euclidean distance is the one that is the most accurate, that is, has the set of performance bottlenecks that is most similar to those of the `reference` input set. (It is important to note that we verified that using ranks did not significantly distort the results as compared to using the PB magnitudes. Rather, using ranks prevented some bottlenecks from dominating the results, which allowed fewer significant bottlenecks to have some limited effect. The reason that we chose ranks over the PB magnitudes was because comparison of ranks is more straightforward and intuitive.)

Finally, our set of processor and memory parameter values is similar to those found in [20].

4.2 Execution-Profile Characterization

If the PB design is a hardware-level characterization, then its software-level counterpart is the basic block characterization. We characterize the basic blocks based on their execution frequencies (BBEF) and their instruction counts (BBV in SimPoint terminology). In this paper, we define a basic block to be the group of instructions between a branch target (taken or not taken) up to the next branch. The BBEF is simply the number of times that each basic block is executed. By comparing the BBEF profiles for the `reference` input set and each technique, we can determine how accurate that technique is in terms of code coverage. The BBV is similar to BBEF except that, instead of incrementing the count by one each time a basic block executes, we increment that basic block's counter by the number of instructions that were executed in that instance of that basic block, which factors in the number of instructions in each basic block.

We use a χ^2 test [14] to compare the distributions of the `reference` input set and each technique. If the χ^2 test value is smaller than the χ^2 statistic, then the two distributions are considered to be statistically similar. We also use the χ^2 test value as a measure of the distance between the two distributions: Similar distributions will have a very small χ^2 test value.

4.3 Architectural-Level Characterization

The last characterization method that we used to compare techniques is at the architectural level. We first vectorize a set of metrics—instructions per cycle (IPC), branch prediction accuracy, level-1 (L1) D-cache hit rate, L1 I-cache hit rate, and level-2 (L2) cache hit rate—after normalizing each metric to its maximum possible value to allow for cross-metric comparisons and then calculate its euclidean distance from the `reference` input set. We included this characterization since these metrics are often used by architects to evaluate their enhancements. However, the principal deficiency of using architectural-level metrics is that, since they average the effect of all factors over time to produce a single number, the effects of larger interactions may counterbalance each other while obscuring the effects of lower order interactions. Table 3 lists the key parameter values for the four configurations used for the architectural-level characterization. These parameter values were chosen based on a survey of several commercial processors.

5 Results of Characterization Methods

The next two sections present the results of our analysis of the accuracy and simulation speed for the seven techniques that were described in Section 2. Section 5 presents the results for the three characterization methods that were described in Section 4, while Section 6 quantifies the SvAT and the potential configuration dependence that each of these techniques have.

5.1 Processor Bottleneck Characterization Results and Analysis

Since the number of elements in each vector of ranks is 43 and since the value of each element is a number between 1 and 43, the maximum euclidean distance between two vectors occurs when the ranks for the two vectors are completely "out of phase," that is, $< 43, 42, 41, \ldots 3, 2, 1 >$ versus $< 1, 2, 3, \ldots 41, 42, 43 >$). This distance is 162.75. Fig. 1 presents the average (mean) euclidean distance away

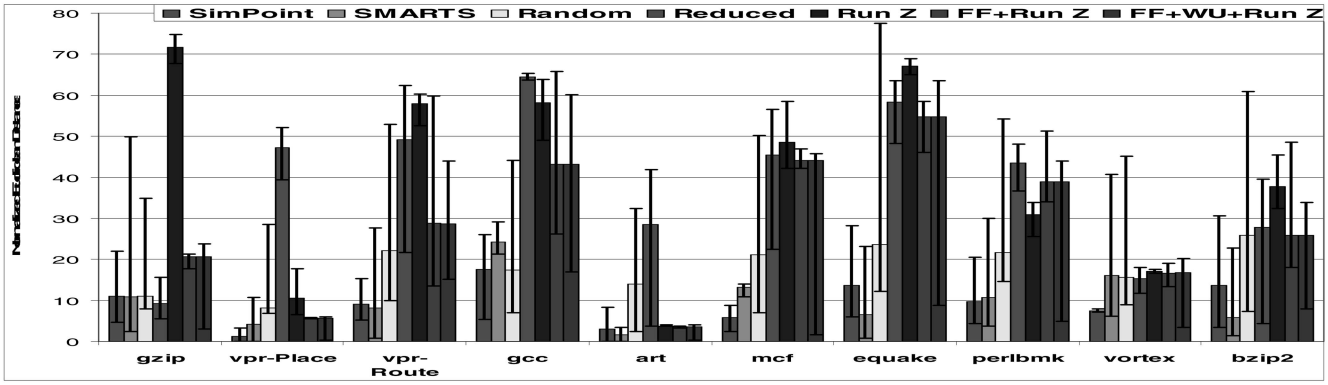

Fig. 1. The normalized euclidean distance away from the `reference` input set for each type of simulation technique for the performance bottleneck characterization. For each technique, the average distance across all permutations is shown, along with the minimum and maximum distance (error bars).

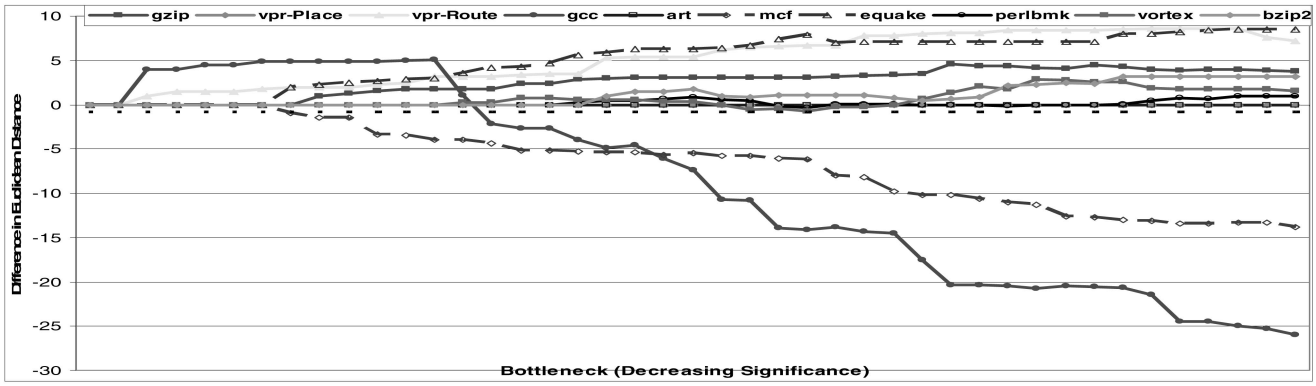

Fig. 2. Difference in the SimPoint and SMARTS euclidean distances in ascending order of `reference` rank.

from the `reference` input set across all permutations for each technique, which is normalized to the maximum distance and scaled to 100.

Across all benchmarks, the accuracy of the reduced input sets varies significantly. In general, the poor accuracy (large euclidean distance) of the reduced input sets is due to two reasons. First, especially in the case of *mcf*, the percentage of cycles due to cache misses serviced by main memory is much larger for the `reference` input set than in any of the reduced input sets. Consequently, we expect (and find) that the reduced input sets tend to underestimate the rank of the memory-hierarchy-related bottlenecks. For example, in *gcc*, the rank of the memory latency for the `reference` input set is 3, whereas its rank for the SPEC `test` reduced input set is 41. Second, our results for the basic block analysis, presented in Section 5.2, shows that the execution profiles of the reduced input sets and the `reference` input set are very different. In other words, using a reduced input set effectively simulates a different program than when using the `reference` input set.

With the exceptions of *vpr-Place* and *art*, the accuracy of the truncated execution techniques is also quite poor. Although the distances for FF X + Run Z and FF X + WU Y + Run Z are lower than the distances for Run Z, the reasons for the poor accuracy of these techniques are the same. First, since computer architects choose the values of X, Y, and Z arbitrarily, these three techniques end up simulating a portion of the program that not only may be uninteresting but may also not be representative of the entire benchmark. Second, given the highly complex phase behavior of some of these benchmarks (*gcc* is an excellent example), simulating a few billion instructions, even after fast-forwarding through a few billion instructions, does not simulate enough phases of the program to elicit a similar set of performance bottlenecks. However, increasing the period of detailed simulation reduces the appeal of this class of techniques by increasing its simulation time.

Of the three sampling-based simulation techniques, SimPoint and SMARTS are significantly more accurate than the reduced input set and truncated execution techniques, whereas random sampling is generally more accurate. Therefore, from a performance bottleneck point of view, the sampling-based techniques have performance bottlenecks that are more similar to the performance bottlenecks induced by the `reference` input set. Overall, SMARTS is slightly more accurate than SimPoint since, for six of the 10 benchmarks, the minimum distance for SMARTS is lower than the minimum distance for SimPoint (in terms of the average distance, SMARTS is smaller for five benchmarks), whereas both are more accurate than random sampling.

It is important to note that large differences in the ranks for bottlenecks that are not significant can increase the apparent distance for a technique. To examine if this is the case, Fig. 2 shows the difference in the SimPoint and SMARTS distances, each with respect to the `reference` input set, that is,

$$\|\text{SimPoint} - \texttt{reference}\| - \|\text{SMARTS} - \texttt{reference}\|,$$

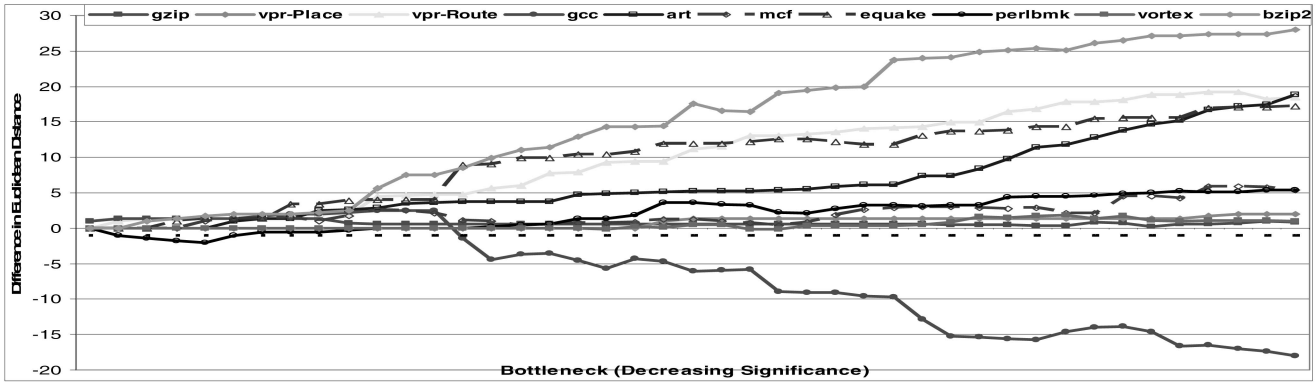

Fig. 3. Difference in the random sampling and SMARTS euclidean distances in ascending order of `reference` rank.

whereas Fig. 3 shows the corresponding comparison between random sampling and SMARTS. Both figures only show the results for the most accurate (smallest euclidean distance) permutation of each technique.

The bottlenecks along the x-axis are sorted in ascending order (the most significant to the least significant) of the `reference` input set rank for *each benchmark*. Therefore, the same rank in different benchmarks may correspond to different bottlenecks. This plotting allows us to examine the effect of each bottleneck in decreasing order of significance in each benchmark. The difference in the distances for bottleneck N is the difference in the distances when only the N most significant bottlenecks are included in the distance calculations. Note that, since the most significant bottlenecks are shown on the left-hand side, larger differences in the euclidean distance on the left-hand side are more significant than the same distance on the right-hand side.

Fig. 2 shows that, for all benchmarks except for *gcc*, there is relatively little difference between the euclidean distances for SimPoint and SMARTS, at least for the most significant bottlenecks. Therefore, we conclude that, for these benchmarks, with the exception of *mcf*, SMARTS is slightly more accurate than SimPoint. For *mcf*, SimPoint is slightly more accurate than SMARTS. For *gcc*, there is a difference in the euclidean distances starting at bottleneck 3 (memory latency) due to a combination of two factors. First, the warm-up approach that we used in this paper (assuming cache hit for each first cache way access) tends to overestimate the cache hit rate, which makes the memory latency appear to be a less significant performance bottleneck than it is. Note that this problem is more severe for benchmarks with large numbers of phases and that these results illustrate the importance of using accurate warm-up techniques. Second, since *gcc* has a very complex phase behavior and, for this specific SimPoint configuration (multiple 10 M simulation points), phase transitions are typically not chosen to be simulation points, which subsequently underestimates the effect of the memory latency. Increasing the maximum number of simulation points, for example, using 1 M simulation points, with a max_K of 300, can minimize this problem. Note, however, that, although the absolute error may be significant, the relative error of SimPoint is constant and relatively small [15].

For *gcc*, it is also important to reiterate that, although the performance bottlenecks, taken as a whole, for SimPoint are more similar to those of the `reference` input set than those for SMARTS, this result is due to the fact that SMARTS is not as accurate as SimPoint in estimating the importance of less significant performance bottlenecks.

Fig. 3 confirms the conclusion that SMARTS is more accurate than random sampling, with the possible exceptions of *gcc* and *perlbmk*. Although random sampling seems to be more accurate than SMARTS for *gcc*, as was the case in Fig. 2, the apparent inaccuracy of SMARTS is solely due to less significant performance bottlenecks. For *perlbmk*, random sampling more accurately estimates the importance of the second, third, fourth, and fifth most significant performance bottlenecks (ROB size, L2 cache latency, type of branch predictor, that is, branch prediction accuracy, and number of integer ALUs, respectively). However, since the PB magnitudes of the second and third most significant bottlenecks are very close (as are the PB magnitudes for the fourth and fifth most significant bottlenecks), the difference in euclidean distances is a little misleading due to the quantization error incurred by using ranks. Nevertheless, random sampling estimates the significance of these four bottlenecks a little more accurately than SMARTS does.

In conclusion, the results in this section show that the reduced input set and truncated execution techniques are very inaccurate as compared to the results obtained by the `reference` input set. By contrast, the three sampling-based techniques—SMARTS, SimPoint, and random sampling, in descending order of accuracy—are significantly more accurate.

5.2 Execution-Profile Characterization Results and Analysis

In this section, we examine how the seven reduced time simulation techniques compare to the `reference` input set when using the execution-profile characterization. Fig. 4 presents the results of this characterization. Due to space limitations, since the results of the BBEF and BBV are virtually identical, we discuss only the results of the BBEF characterization. Additionally, we also present the results for *gzip*, *gcc*, *mcf*, and *bzip2* only. Note, however, that the results of these benchmarks are representative of the entire set.

As described in Section 2, if the variability in the CPI between sampling units is too high, which depends on the specific processor configuration being simulated, SMARTS recommends a higher sampling frequency. However, since the execution profile is independent of the processor

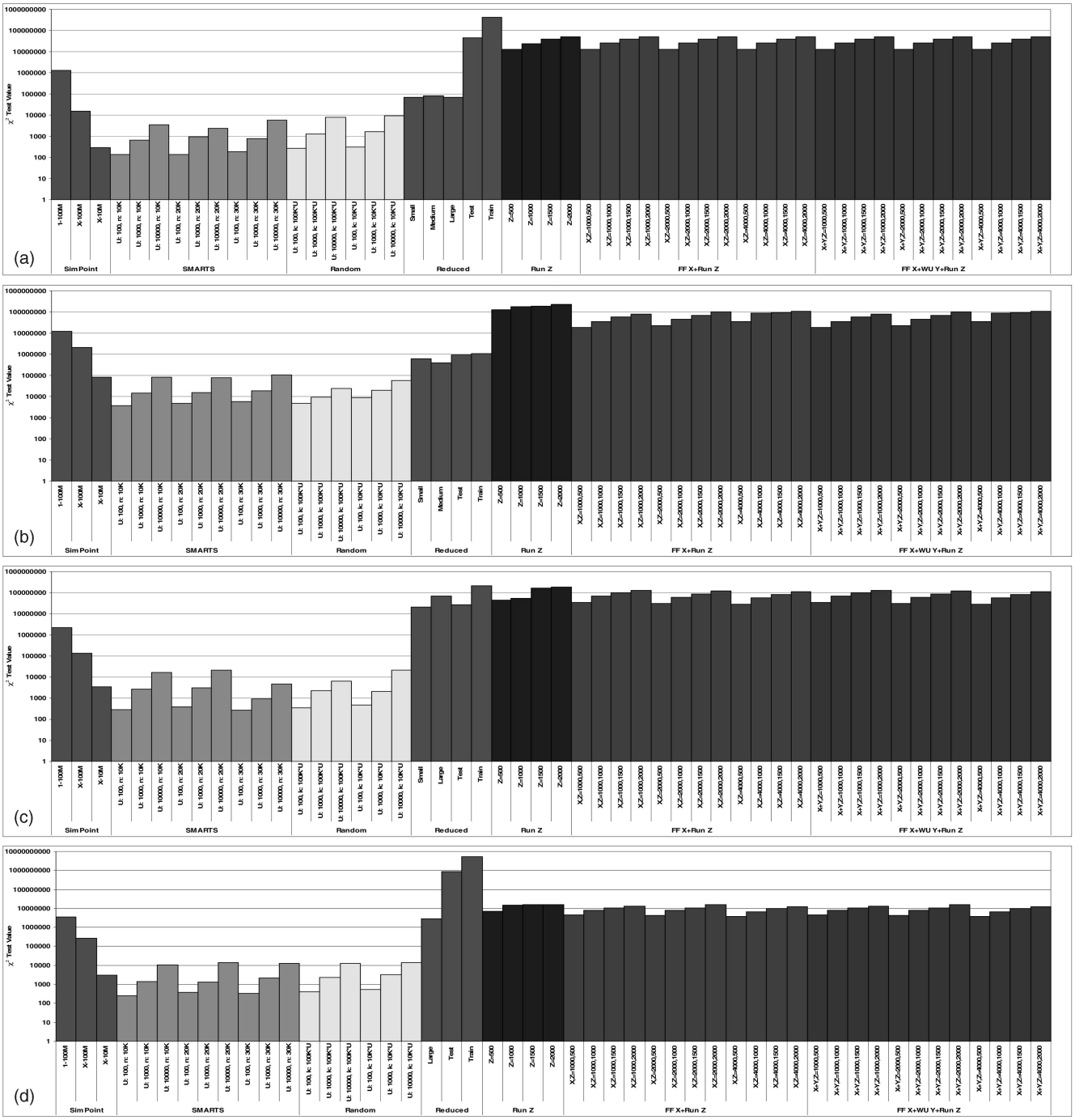

Fig. 4. BBEF execution profile. Results are in terms of the χ^2 distance. (a) *gzip*. (b) *gcc*. (c) *mcf*. (d) *bzip2*.

configuration, for SMARTS, we measure the execution profile for several different initial sampling frequencies (10,000, 20,000, and 30,000).

In the same vein, since the processor warm-up does not affect the execution profile, we ignore the permutations that differ only in their warm-up lengths. For example, for SMARTS, U: 100, W: 200 versus U: 100, W: 2,000 versus U: 100, W: 20,000, and $\mathrm{FF}\ 999\mathrm{M} + \mathrm{WU}\ 1\mathrm{M} + \mathrm{Run}\ 500\mathrm{M}$ versus $\mathrm{FF}\ 990\mathrm{M} + \mathrm{WU}\ 10\mathrm{M} + \mathrm{Run}\ 500\mathrm{M}$ versus $\mathrm{FF}\ 900\mathrm{M} + \mathrm{WU}\ 100\mathrm{M} + \mathrm{Run}\ 500\mathrm{M}$.

In Fig. 4, the y-axis corresponds to the χ^2 test value between the basic block distribution of each simulation technique and the corresponding distribution for the `reference` input set. Therefore, if a technique has a very similar execution profile (each basic block executes the same fraction of the time), the χ^2 distance will be very small. Note that the scale of the y-axis is logarithmic.

The results in Fig. 4 show that the execution profiles of the three sampling-based techniques are more similar to the execution profile of the `reference` input set than the reduced input set and truncated execution techniques. In other words, the sampling-based techniques execute the same basic blocks and at more similar frequencies than the other four techniques.

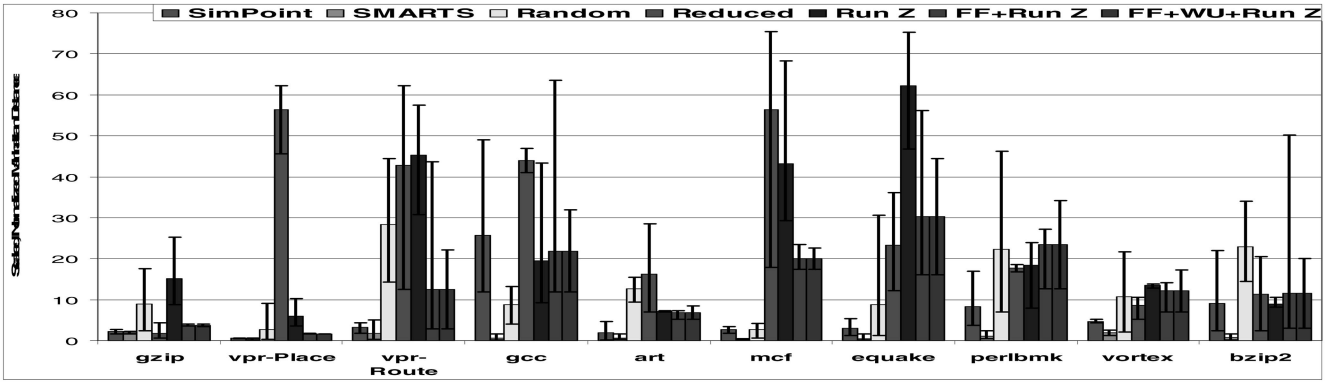

Fig. 5. Accuracy of architectural metrics in terms of their scaled normalized Manhattan distance for configuration 1.

Given the fact that the truncated execution techniques execute a single, essentially random, interval of instructions, it is not surprising to see that their execution profiles for all benchmarks are extremely dissimilar as compared to the execution profile of the `reference` input set. For a similar reason, given that the reduced input sets may simulate a totally different input than the `reference` input set, it is also not surprising to find that their execution profiles differ. However, for some benchmarks, if the reduced input set is substantially similar to the `reference` input set, then the execution profile of that reduced input set may be very similar to that of the `reference` input set. For example, the MinneSPEC reduced input sets of *gzip* are simply a truncated version of the `reference` input set.

Of the three sampling-based techniques, SMARTS and random sampling have execution profiles that are very similar to the execution profile of the `reference` input set, followed by SimPoint. The reason that SMARTS and random sampling are more accurate than SimPoint from an execution-profile point of view is that SimPoint does not touch some basic blocks at all, whereas the probability of touching a basic block with SMARTS and random sampling is proportional to the frequency of the execution of each block. This is the same reason that the X-10,000,000 permutation is more accurate than either the 1–100,000,000 or X-100,000,000 permutations. The X-10,000,000 permutation simply touches more basic blocks. Finally, the results for SimPoint show that, although 1-100,000,000 does not have a very similar execution profile, SimPoint is significantly better at picking a single representative interval, from a code perspective, as compared to truncated execution.

Finally, the results show that, at a 95 percent confidence level, the execution profiles of all permutations of the reduced input sets and the truncated execution techniques are statistically different from the execution profile of the `reference` input set. This conclusion is not surprising given the nature of these simulation techniques and the results in the previous section.

On the other hand, the execution profiles of the three sampling-based techniques are statistically similar to the execution profile of the `reference` input set for most permutations, especially for the ones that have the largest number of sampling units, for example, X-10,000,000 for SimPoint and U: 100 for SMARTS/random sampling. Obviously, if a permutation of a sampling-based technique has a larger number of sampling units, the sampling units are more spread out across the benchmark, which makes it more likely that all basic blocks will be touched at the correct relative frequency.

(Due to improperly scaling the basic block vectors, our previous work [21] showed that the execution profiles of all techniques were statistically similar to that of the `reference` input set. These new results correct that previous error.)

5.3 Architectural-Level Characterization Results and Analysis

While the results in the previous two subsections quantified the accuracy of each of the simulation techniques from a performance bottleneck and execution-profile point of view, what is more important is that each technique has similar architectural-level metrics (IPC, branch prediction accuracy, L1 D-cache hit rate, L2 cache hit rate, and L1 I-cache hit rate) as the `reference` input set. More specifically, after extracting the metrics, we

1. normalize the value of each metric to its maximum value,
2. scale the normalized value to 100,
3. vectorize the scaled normalized values, and
4. compute the Manhattan distance between each vector for each technique and the vector for the `reference` input set.

Since there is no defined certain maximum for CPI, we use IPC instead, which has a maximum value equal to the issue/commit width. We use the Manhattan distance instead of the euclidean distance because the difference in the distance is equal to the difference in the scaled normalized values of each metric.

Fig. 5 presents the architectural-level accuracy for each of the reduced time simulation techniques for configuration 1. The results for the other three configurations were similar. The y-axis shows the scaled normalized Manhattan distance between the vectors of architectural metrics for each technique and the `reference` input set. The height of each bar shows the average accuracy in terms of the Manhattan distance for that technique, whereas the error bars show the minimum and maximum distances.

The results in Fig. 5 are very similar to those in Fig. 1. More specifically, Fig. 5 shows that the reduced input sets and truncated execution techniques are not very accurate, whereas the three sampling-based techniques are significantly more accurate: SMARTS is the most accurate, followed by SimPoint, and then random sampling. Note

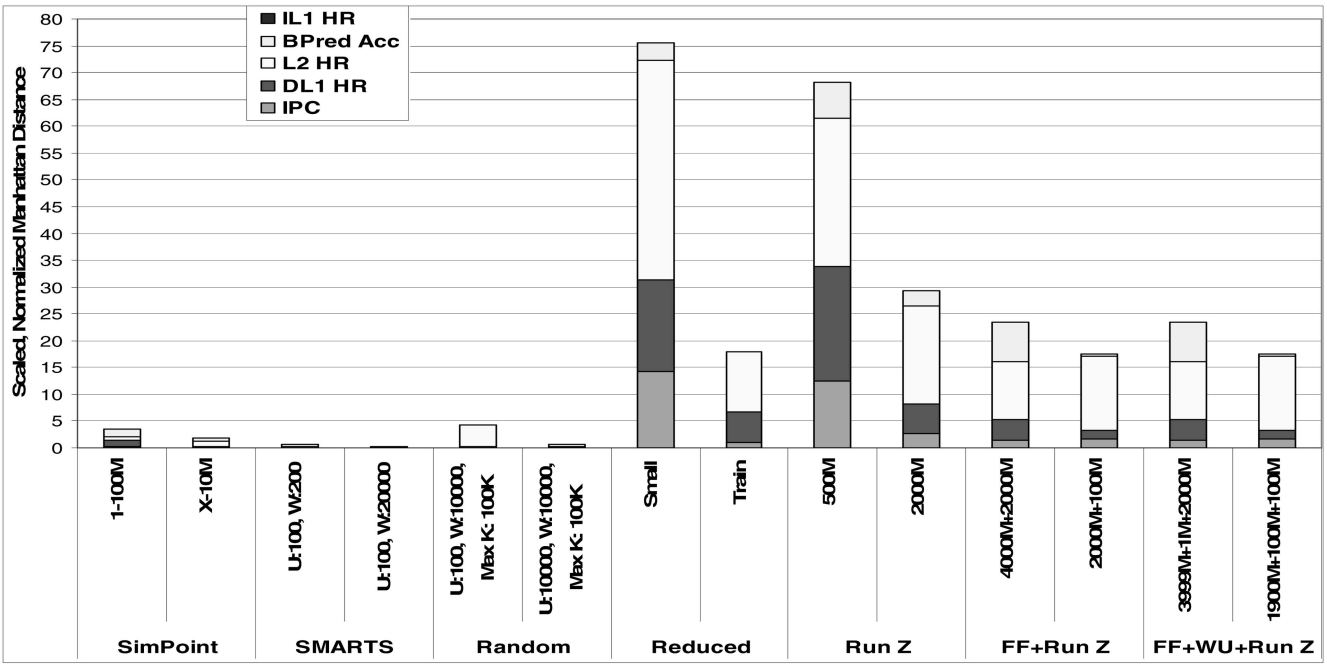

Fig. 6. Accuracy of each architectural metric in terms of their scaled normalized Manhattan distance for configuration 1 for *mcf*.

that the large error in the SimPoint results for *gcc* is due solely to the higher-than-expected cache hits rates from using the assume-cache-hit warm-up technique.

Fig. 6 shows the accuracy of each architectural metric for the worst (left) and best (right) permutations of each technique for *mcf*. A permutation was the worst (best) if it had the largest (smallest) overall Manhattan distance. Note that the height of each bar in Fig. 6 corresponds to the end point location of the error bars for each technique in Fig. 5.

Fig. 6 shows that the poor accuracy of the reduced input set and truncated execution techniques is largely due to large differences in the L1 D-cache and L2 cache miss rates, which, for this configuration with the `reference` input set, are 30.2 percent and 63.1 percent, respectively. By contrast, the corresponding miss rates of the `small` reduced input set, for example, are 13.0 percent and 22.1 percent, respectively. These results are not particularly surprising, given that the reduced input set techniques do not simulate enough instructions (and, therefore, do not generate enough misses) and that the truncated execution techniques have significantly smaller effective working set sizes.

On the other hand, the memory and, correspondingly, overall behavior of the three sampling-based techniques are significantly more similar to that of the `reference` input set. Comparing the two random results, we see that the difference between the worst and the best permutations is only in the number of instructions in each sampling unit. The improvement in the L2 cache hit rate distance is due to the "self-warming" effect that the larger sampling unit has.

5.4 The Efficacy of the Characterization Methods

It is extremely important to note that, since these three characterizations examine the accuracy of the seven reduced time simulation techniques from three different perspectives, the coherency of the results and conclusions indicates that the accuracy of each technique is not merely a side effect or construct of the specific characterization method, but, rather, an intrinsic property of the technique. Therefore, although the conclusions are the same for all three characterizations, the coherency across all three bolsters the validity of the conclusions and the efficacy of the characterizations.

6 An Analysis of the SvAT and Potential Configuration Dependence

6.1 SvAT Analysis

Computer architects typically assume that increased simulation speed comes at the cost of reduced simulation accuracy such that the ideal technique minimizes the loss of accuracy while maximizing the simulation speed. Although accuracy is the preeminent characteristic, speed emerges as an important consideration when the accuracies of several techniques are similar.

To accurately determine the SvAT of these techniques, for all seven techniques, we simulated their permutations on the same machine to eliminate any differences in the processor, memory subsystem, network, operating system, and so forth. Although we ran each test case only once (due to the number of simulations), since we simulated almost 50 processor configurations for each permutation, the amount of experimental error was negligible. The processor configurations represent the envelope of the hypercube of potential configurations.

Fig. 7 presents the SvAT graphs for all benchmarks. Speed and accuracy are on the x and y-axes, respectively. The speed of a technique is simply the total simulation time of that technique as a percentage of the total simulation time of the `reference` input set, whereas the accuracy of that technique is the Manhattan distance between the CPI vectors of the technique and the `reference` input set. (We used the Manhattan distance instead of the euclidean distance in this analysis because it presented the results more clearly.) We included the cost of generating simulation points (for

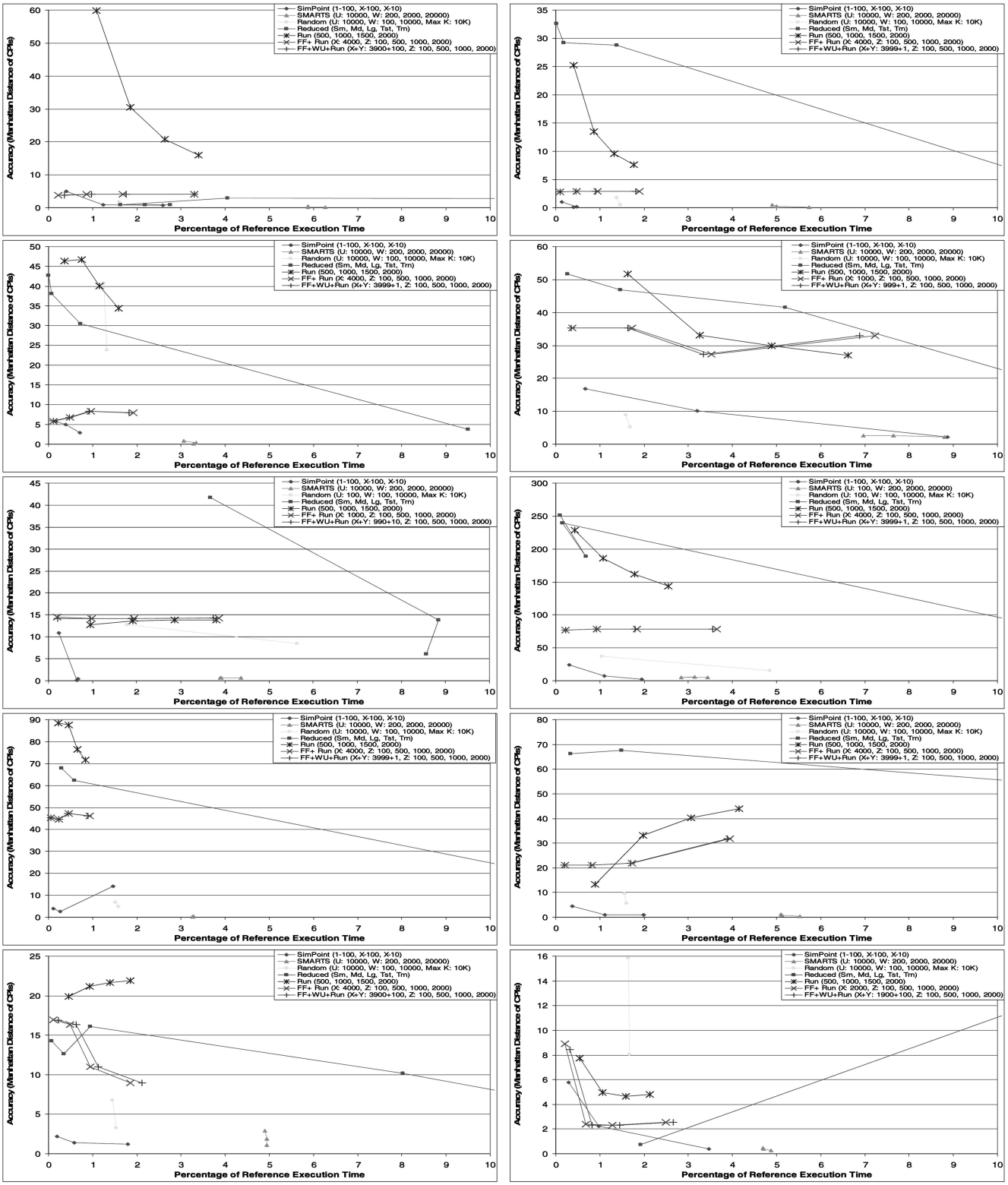

Fig. 7. Simulation speed versus accuracy trade-off. The graphs, clockwise from the top left, are for *gzip*, *vpr-Place*, *gcc*, *mcf*, *perlbmk*, *bzip2*, *vortex*, *equake*, *art*, and *vpr-Route*.

SimPoint) and simulation checkpoints (for SimPoint and the truncated execution techniques) into the simulation speed. (Versions of SimPoint newer than SimPoint 1.0 dramatically reduce the time needed to determine the simulation points, but were not available when we started this study.) The costs of generating the reduced input sets and the initial profiling of SMARTS were not included as these costs were not quantified in [13] and [18], respectively. However, for SMARTS, the simulation times of the simulations that did not sample at a high enough frequency, that is, the required additional simulations, were included in the cost. (Across all benchmarks, the average number of

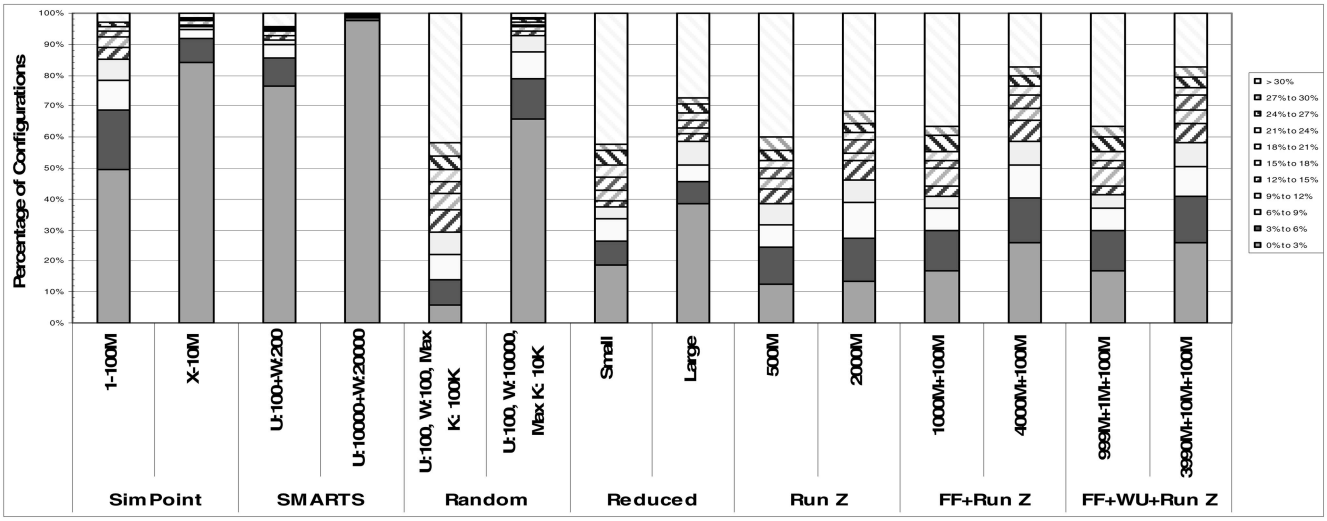

Fig. 8. Configuration dependence: histogram of CPI error (relative to `reference`) for all benchmarks.

simulations for each SMARTS permutation ranged from 1.00 to 1.59, with a maximum of 6. Using slightly higher-than-recommended sampling frequencies can reduce the maximum number of simulations to about 3 [19].) Including or excluding the costs of a technique merely moves/stretches the technique's lines right or left, respectively. Finally, to reduce the clutter of each graph, we only present the highest accuracy permutation of the SMARTS, random sampling, FF X + Run Z, and FF X + WU Y + Run Z techniques. The legend specifies the exact permutation for each technique.

The first key conclusion that we draw from these figures is that the SvAT of the reduced input set and the truncated execution techniques is very poor. Not only is their accuracy very poor, but also their poor accuracy is compounded by simulation times that are either not inversely proportional to their accuracy (that is, poor accuracy and speed) or significantly higher than for the other techniques. In particular, the `train` input set has the worst SvAT since its accuracy ranks toward the bottom and since its simulation time is significantly longer than any other technique. Therefore, the reduced input set and truncated execution techniques, from the dual viewpoints of simulation accuracy and speed, do not offer any advantages as compared to SimPoint, SMARTS, and random sampling.

It is interesting to note that increasing the detailed simulation period of the truncated execution techniques does not automatically confer a higher accuracy, for example, *gcc* and *perlbmk*. Rather, naively increasing the simulation period can simultaneously decrease both the simulation accuracy and speed.

Although the accuracy of SimPoint is not quite as good as SMARTS, SimPoint has a better SvAT than SMARTS does, even after including the cost of generating the simulation points (which is zero if the architect uses those found on the SimPoint Web page) and including the cost of generating the checkpoints (the cost of which is amortized by successive runs and can be decreased by picking early simulation points [15]). Therefore, if the architect's principal concern is accuracy, then SMARTS is the most appropriate technique. However, if the architect is willing to sacrifice a little accuracy for an increased simulation speed (and who is not, around deadline time), then SimPoint is the most appropriate technique.

As shown in Section 5, the accuracy of random sampling is slightly worse than the accuracy of SMARTS and SimPoint, but, as shown in Fig. 7, its simulation speed is typically between the simulation speeds of SimPoint and SMARTS. Therefore, from the results in Fig. 5, we conclude that the SvAT of random sampling is not as good as SimPoint (due to slightly lower accuracy and speed) or SMARTS (due to a significantly lower accuracy).

In summary, from the perspective of an SvAT, the best techniques are, listed in descending order of their SvAT, SimPoint, SMARTS, random sampling, FF X + Run Z, FF X + WU Y + Run Z, Run Z, and reduced input sets, although there is a large separation between the three sampling-based techniques and the others.

6.2 Potential Configuration Dependence

Another relevant consideration for computer architects is how the accuracy of these techniques changes based on the processor configuration. The absolute accuracy of the ideal technique will remain constant across a broad range of configurations. A predictable and stable accuracy allows trends to emerge from the noise of error.

To quantify the magnitude of this potential problem, we calculated the percentage error between the CPIs of each technique and the `reference` input set and then determined the frequencies of the CPI error for all configurations. A technique is configuration dependent when the magnitude of the CPI error varies significantly across configurations. Another way of quantifying the configuration dependence is to calculate the standard deviation of the CPI error. A high standard deviation means that the CPI error is very different across configurations, which means that that technique is configuration dependent. Fig. 8 shows the percentage of configurations that fell into each range of CPI errors for that specific permutation across all benchmarks. For each technique, Fig. 8 shows the worst (left) and best (right) permutations. A permutation was selected as

the worst or best when it had the highest or lowest standard deviation for its CPI error.

Fig. 8 shows four key results. First, even for the best permutations, we conclude that the reduced input set and truncated execution techniques have a significant configuration dependence because the CPI error is distributed across multiple error ranges. Second, SMARTS has very little, if any, configuration dependence since the CPI error of most configurations falls into one error range, that is, the 0 percent to 3 percent one. Even for the worst permutation, more than 75 percent of its configurations have a CPI value that is within 3 percent of the CPI for the `reference` input set. In the best permutation, this percentage climbs to almost 98 percent, which is slightly less than the target of 99.7 percent of the configurations being within ± 3 percent of the `reference` input set's CPI [18]. Nevertheless, given the "extreme" nature of the configurations, that is, they represent configurations at the envelope of the hypercube of potential configurations and, in essence, constitute a "stress test" for the reduced time simulation techniques, we conclude that SMARTS has virtually no configuration dependence. Third, for SimPoint, in the worst permutation, there is a significant configuration dependence that largely disappears in the best permutation. However, even in the best permutation, the percentage of configurations for which the CPI error is greater than 3 percent is lower than the best case for SMARTS. Fourth and finally, like SimPoint, random sampling has a significant configuration dependence for the worst permutation, which clearly diminishes for the best permutation. However, overall, random sampling is more significantly configuration dependent as compared to SimPoint.

Although the results in Fig. 8 show the frequency of CPI errors across all benchmarks, we found that the results presented in Fig. 8 were fairly typical for each benchmark and that no benchmarks were "outliers" in terms of their frequency of CPI error.

In conclusion, the results in this section show that the reduced input set and truncated execution techniques are severely configuration dependent because their CPI results are very inaccurate and the CPI error does not trend. By contrast, SimPoint and SMARTS have very little, if any, configuration dependence because the CPI error is generally small and consistent. Finally, the configuration dependence for random sampling ranges from severe to good for the worst and best permutations, respectively.

7 Related Work

Although we found several papers that were somewhat related to this paper, we did not find any papers that comprehensively evaluated the accuracy of all techniques and found only one [10] that independently evaluated more than one technique. Most papers that evaluated the accuracy of techniques did so in the context of comparing the results of a new technique to the results when using the `reference` input set.

In [10], Gómez et al. show that the L2 behavior and branch prediction accuracy of the MinneSPEC reduced input sets are not `reference`-like and that the accuracy of truncated execution depends heavily on fast-forwarding to a representative interval. In contrast to the results presented in this paper, they do not evaluate any sampling-based techniques and only evaluate a limited number of permutations for the reduced input sets and truncated execution. Furthermore, their results did not analyze the accuracy of the techniques by using multiple characterizations nor did they evaluate the SvAT and the potential configuration dependence.

Outside of [10], the most relevant related work falls into two categories: simulation methodology and simulator validation.

Simulation methodology. Yi et al. [20] proposed using a PB design as a means of introducing a statistical rigor into the simulation methodology. More specifically, they used a PB design to identify the most significant bottlenecks to help choose parameter values, to select a statistically different set of benchmarks, and to measure the effect that an enhancement has on the processor. The first two applications attempt to improve the simulation setup phase, whereas the last application improves the analysis phase.

Eeckhout et al. [8] used statistical data analysis techniques to determine the statistical similarity of benchmark and input set pairs. To quantify the similarity, they used metrics such as instruction mix, branch prediction accuracy, cache miss rates, number of instructions in a basic block, and maximum amount of parallelism inherent to the benchmark. After characterizing each benchmark with these metrics, they used statistical techniques such as principal component and cluster analyses to cluster the benchmarks and input set pairs together.

Simulator validation. Black and Shen [1] iteratively improved the accuracy of their performance model by comparing the cycle count of their simulator, which targeted a specific architecture, against the cycle count of the actual hardware. Their results show that modeling, specification, and abstraction errors were still present in their simulation model, even after a long period of debugging. Their work showed the need for an extensive iterative validation before the results from a performance model can be trusted.

Desikan et al. [7] measured the amount of error, as compared to the Alpha 21264 processor, which was present in an Alpha version of the SimpleScalar simulator. Their results showed that the simulators that model a specific processor, such as SimpleScalar, generally report higher IPCs than simulators that are validated against a real machine. On the other hand, unvalidated simulators that targeted a specific machine underestimated the performance.

Gibson et al. [9] described the types of errors that were present in the FLASH simulator when compared to the custom-built FLASH multiprocessor system. To determine which errors were present in the FLASH simulator, they compared the simulated execution time from the FLASH simulator against the actual execution time of the FLASH processor. Their results showed that the margin of error (the percentage difference in the execution time) of some simulators was more than 30 percent, which is higher than the speedups that are often reported for specific architectural enhancements.

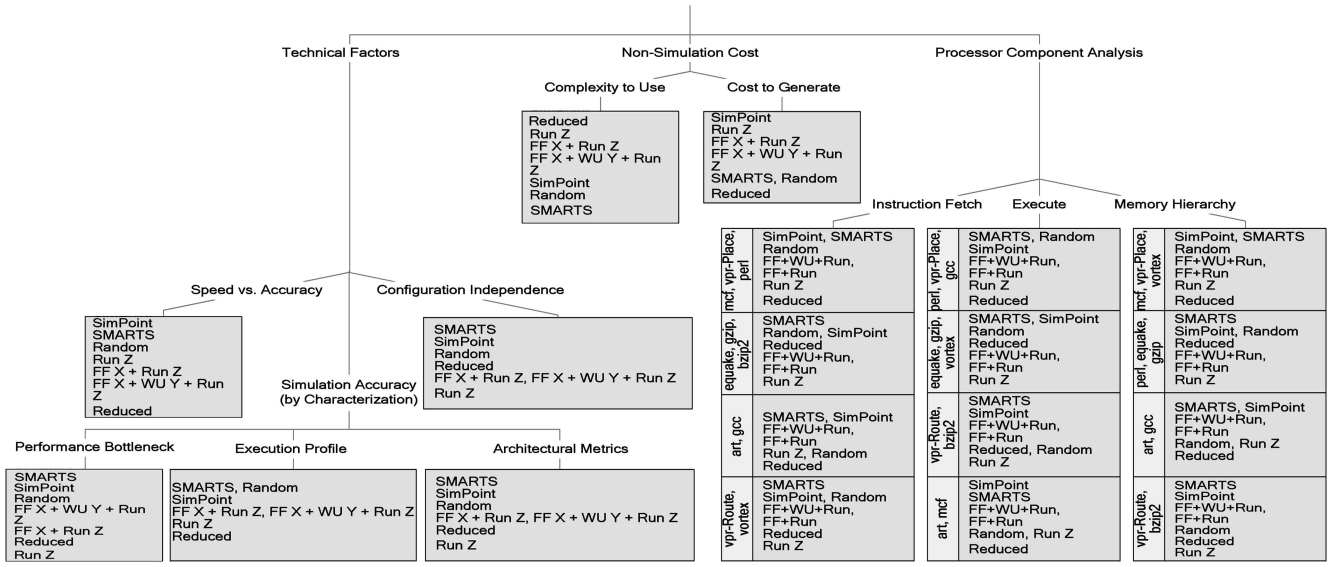

Fig. 9. Decision tree for the selection of a simulation technique.

8 Recommendations

Based on the results of the three characterization methods, the SvAT, and the configuration dependence analysis, we make three recommendations for performing simulation-based architecture studies.

Recommendation 1: Improve the documentation of simulation methodologies. From our survey of simulation methodologies, the number of unknown techniques accounted for half of all papers from HPCA, ISCA, and MICRO 1994 to 2003, as well as approximately one-third of the papers in the later years. Inadequately documenting how the results were obtained prevents other researchers from verifying those results or building upon them. More importantly, results that are presented without adequate documentation or justification of the simulation methodology may be considered to be suspect.

Recommendation 2: Sampling-based simulation techniques should be used when the goal is to get reference-like results. Simulation with reduced input sets should be viewed as using a completely different benchmark program than what is obtained when using the `reference` input set. Given its generally low level of accuracy, the truncated execution technique should not be used since any conclusions that are drawn from the results by using this technique may simply be a figment of the technique rather than a bona fide effect. Due to the very high levels of accuracy and their very low simulation times, we highly recommend that sampling-based techniques, as epitomized by SimPoint, SMARTS, and random sampling, be used instead. Although this may seem to be an intuitively obvious recommendation, the fraction of papers that used the reduced input sets or truncated execution techniques actually increased from 68.9 percent in the eight years prior to the introduction of SimPoint to 82.1 percent in the conferences that occurred after SimPoint was introduced. Finally, benchmarks from old benchmark suites should not be used unless there is a compelling reason to do so, especially so since SimPoint and SMARTS are both fast and accurate. In our survey, we found a surprising number of papers that used benchmarks that were more than five years old. (So as to not sound too preachy, we would like to point out that we have been guilty of some of these problems ourselves.)

Recommendation 3: Suggestions for selecting a simulation technique. Based on the results presented in the previous two sections and from our experience in this study, Fig. 9 presents the detailed ordering of the seven techniques for several different categories. The *Technical Factors* branch orders the techniques based on the conclusions from the three characterizations (performance bottleneck, execution profile, and architectural level), the SvAT, and the configuration dependence analysis, the results of which were presented in Sections 5 and 6.

The *Complexity to Use* category reflects the complexity of the changes to the simulator that are needed to support that technique. Since the reduced input sets do not require any changes, they have the lowest complexity to use. SMARTS and random sampling have the highest complexity to use since they require the largest number of changes to the simulator, such as support for periodic/random sampling, functional warming, and statistical calculations. The other four techniques have a medium complexity to use because they could require changes to the simulator to support fast-forwarding, warm-up, and early termination.

The *Cost to Generate* category is the amount of effort that is needed to "create" each technique. Since SimPoint requires minimal user intervention to find a benchmark's simulation points, it has the lowest cost. Note, however, that, for some compiler-based studies, the architect may need to repeatedly generate new simulation points to reflect the status of various levels of code optimization. On the other end of the spectrum, SMARTS, random sampling, and reduced input sets have the highest costs to generate since new SMARTS and random sampling parameters (U, W) may need to be found or new reduced input sets need to be created for each benchmark suite. The truncated execution techniques may require a moderate cost to generate if the

architect does a little bit of profiling to determine reasonable fast-forwarding, warm-up, and run values.

The *Processor Component Analysis* branch orders the techniques based on how similar their performance bottlenecks are within each of the processor's major components, that is, Instruction Fetch, Execute, and Memory Hierarchy. After eliminating the bottlenecks that are less significant than the dummy bottlenecks (that is, noise), we then compute the euclidean distance between each technique and the `reference` input set. Then, to facilitate comparisons across techniques and components, we divide the euclidean distance by the number of significant bottlenecks in the `reference` input set. By focusing on specific components, this analysis determines which techniques should be used for which components.

Although these results confirm the results obtained in the previous sections, there are some interesting results. For each of the three components and each benchmark, SMARTS, SimPoint, and random sampling show the best behavior overall. For *gzip*, *equake*, and *vpr-Route*, Run `Z` is not appropriate for any of the components and therefore should not be used. The reduced input sets are not appropriate when focusing on Instruction Fetch and Memory Hierarchy bottlenecks for *gcc* and *vpr-Place*. On the other hand, for *gzip*, the reduced input sets are almost as accurate as SMARTS, SimPoint, and random sampling.

In summary, for different processor components and for a particular benchmark, the accuracy of a technique with respect to the others may change. However, SMARTS, SimPoint, and random sampling are the best overall techniques.

9 Conclusion

With the advent of relatively detailed simulators such as SimpleScalar, simulating the `reference` input set of a SPEC 2000 benchmark to completion is not an option for most computer architects. Consequently, architects have proposed several simulation techniques, such as reduced input sets, truncated execution, and sampling, with the intent of decreasing the simulation time.

We used three characterizations to determine the accuracy of each technique with respect to the `reference` input set. First, we used the statistical PB design to perform a performance bottleneck characterization of each technique. Second, we performed an execution-profile analysis by tallying the BBEF and BBV counts. Third, we compared several architectural performance metrics. After evaluating the accuracy of these techniques with the previous three characterizations, we evaluated the SvAT and the potential configuration dependence of each technique.

Our results lead to several important conclusions. First, the accuracy of the reduced input set and truncated execution techniques was very poor for all three characterizations and the poor accuracy of these four techniques is not offset by a faster simulation speed, which further diminishes their utility. Second, these techniques are significantly configuration dependent because they have a high frequency of large CPI errors and because the CPI error does not trend. Third, our results showed that, for all three characterizations and for the configuration dependence analysis, SimPoint, SMARTS, and random sampling are significantly more accurate techniques. Fourth, although SMARTS is slightly more accurate, SimPoint has a better SvAT. On the other hand, the SvAT of random sampling is not as good as it is for SimPoint and SMARTS due to lower accuracy without a significantly faster simulation speed.

Acknowledgments

This work was supported in part by US National Science Foundation Grant CCF-0541162, Intel, and the Minnesota Supercomputing Institute. A preliminary version of this work was first presented at the 11th International Symposium on High-Performance Computer Architecture (HPCA '05) [21].

References

[1] B. Black and J. Shen, "Calibration of Microprocessor Performance Models," *Computer,* vol. 31, no. 5, pp. 59-65, May 1998.
[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural Level Power Analysis and Optimizations," *Proc. 27th Int'l Symp. Computer Architecture (ISCA '00),* 2000.
[3] D. Citron, "MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences," *30th Int'l Symp. Computer Architecture (ISCA '03) Panel Discussion,* 2003.
[4] T. Conte, M. Hirsch, and K. Menezes, "Reducing State Loss for Effective Trace Sampling of Superscalar Processors," *Proc. Int'l Conf. Computer Design (ICCD),* 1996.
[5] T. Conte and P. Bryan, personal communication, 2005.
[6] T. Conte and P. Bryan, "Statistical Techniques for Processor and Cache Simulation," *Performance Evaluation and Benchmarking,* L.K. John and L. Eeckhout, eds., chapter 6, CRC Press, 2005.
[7] R. Desikan, D. Burger, and S. Keckler, "Measuring Experimental Error in Microprocessor Simulation," *Proc. 28th Int'l Symp. Computer Architecture (ISCA '01),* 2001.
[8] L. Eeckhout, H. Vandierendonck, and K. De Bosschere, "Workload Design: Selecting Representative Program-Input Pairs," *Proc. 11th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT '02),* 2002.
[9] J. Gibson, R. Kunz, M. Ofelt, M. Horowitz, and J. Hennessy, "FLASH vs. (Simulated) FLASH: Closing the Simulation Loop," *Proc. Ninth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '00),* 2000.
[10] I. Gómez, L. Pifiuel, M. Prieto, and F. Tirado, "Analysis of Simulation-Adapted SPEC 2000 Benchmarks," *Computer Architecture News,* vol. 30, no. 4, pp. 4-10, Sept. 2002.
[11] G. Hamerly, E. Perelman, J. Lau, and B. Calder, "SimPoint 3.0: Faster and More Flexible Program Analysis," *J. Instruction Level Parallelism,* Sept. 2005.
[12] J. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *Computer,* vol. 33, no. 7, pp. 28-35, July 2000.
[13] A. KleinOsowski and D. Lilja, "MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research," *IEEE Computer Architecture Letters,* vol. 1, June 2002.
[14] D. Lilja, *Measuring Computer Performance.* Cambridge Univ. Press, 2000.
[15] E. Perelman, G. Hamerly, and B. Calder, "Picking Statistically Valid and Early Simulation Points," *Proc. 12th Int'l Conf. Parallel Architectures and Compilation Techniques (PACT '03),* 2003.
[16] R. Plackett and J. Burman, "The Design of Optimum Multifactorial Experiments," *Biometrika,* vol. 33, no. 4, pp. 305-325, June 1946.
[17] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *Proc. 10th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS '02),* 2002.
[18] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "SMARTS: Accelerating Microarchitectural Simulation via Rigorous Statistical Sampling," *Proc. 13th Int'l Symp. Computer Architecture (ISCA '03),* 2003.
[19] R. Wunderlich, personal communication, 2004.

[20] J. Yi, D. Lilja, and D. Hawkins, "A Statistically-Rigorous Approach for Improving Simulation Methodology," *Proc. Ninth Int'l Symp. High-Performance Computer Architecture (HPCA '03),* 2003.
[21] J. Yi, S. Kodakara, R. Sendag, D. Lilja, and D. Hawkins, "Characterizing and Comparing Prevailing Simulation Techniques," *Proc. 11th Int'l Symp. High-Performance Computer Architecture (HPCA '05),* 2005.

Joshua J. Yi received the BS, MS, and PhD degrees in electrical engineering from the University of Minnesota, Minneapolis. His PhD dissertation focused on measuring and mitigating the performance bottlenecks in processors. He is currently a performance analyst at Freescale Semiconductor, Inc., in Austin in the Great State of Texas. His research interests include high-performance computer architecture, simulation, low-power design, and reliable computing. He is the co-organizer of the Workshop on Modeling, Benchmarking, and Simulation (MoBS) and Workshop on Computer Architecture Research Directions (CARD). He is a member of the IEEE and the IEEE Computer Society.

Resit Sendag received the BS degree in electronics engineering from Hacettepe University, Ankara, Turkey, the MS degree in electrical engineering from Cukurova University, Adana, Turkey, and the PhD degree in electrical engineering, from the University of Minnesota, Minneapolis. He is currently an assistant professor of electrical and computer engineering at the University of Rhode Island, Kingston. His research interests include high-performance computer architecture, memory systems performance issues, and parallel computing. He is a member of the IEEE and the IEEE Computer Society.

David J. Lilja received the BS degree in computer engineering from Iowa State University, Ames, and the MS and PhD degrees in electrical engineering from the University of Illinois, Urbana-Champaign. He is currently a professor and the head of the Department of Electrical and Computer Engineering and a fellow of the Minnesota Supercomputing Institute, University of Minnesota, Minneapolis. He also serves as a member of the graduate faculties in computer science and scientific computation. He has been a visiting senior engineer in the Hardware Performance Analysis Group at IBM, Rochester, Minnesota, and a visiting professor at the University of Western Australia, Perth, in which he was supported by a Fulbright award. Previously, he worked as a research assistant at the Center for Supercomputing Research and Development at the University of Illinois and as a development engineer at Tandem Computers Inc. (now a division of Hewlett-Packard), Cupertino, California. His primary research interests are high-performance computer architecture, parallel computing, hardware-software interactions, nanocomputing, and performance analysis. He has been the chair of or served on the program committees of numerous conferences. He was a distinguished visitor of the IEEE Computer Society, of which he is a member, is a member of the ACM, a fellow of the IEEE, and is a registered professional engineer in electrical engineering in Minnesota and California.

Douglas M. Hawkins received the degree from Witwatersrand University, Johannesburg, South Africa. He is a former chairman of the School of Statistics, Witwatersrand University. He started his own statistical consultancy company and then joined the School of Statistics at the University of Minnesota. His research interests include statistical tools for quality improvement, data diagnostics, data mining, and multivariate methods.