Characterizing and Comparing Prevailing Simulation Techniques

Joshua J. Yi[1], Sreekumar V. Kodakara[2], Resit Sendag[3], David J. Lilja[2], Douglas M. Hawkins[4]

[1] - Networking and Computing Systems Group
Freescale Semiconductor, Inc.
Austin, TX
joshua.yi@freescale.com

[2] - Department of Electrical and Computer Engineering
University of Minnesota
Minneapolis, MN
{sreek, lilja}@ece.umn.edu

[3] - Department of Electrical and Computer Engineering
University of Rhode Island
Kingston, RI
sendag@ele.uri.edu

[4] - School of Statistics
University of Minnesota
Minneapolis, MN
doug@stat.umn.edu

Abstract

Due to the simulation time of the `reference` *input set, architects often use alternative simulation techniques. Although these alternatives reduce the simulation time, what has not been evaluated is their accuracy relative to the* `reference` *input set, and with respect to each other. To rectify this deficiency, this paper uses three methods to characterize the reduced input set, truncated execution, and sampling simulation techniques while also examining their speed versus accuracy trade-off and configuration dependence. Finally, to illustrate the effect that a technique could have on the apparent speedup results, we quantify the speedups obtained with two processor enhancements. The results show that: 1) The accuracy of the truncated execution techniques was poor for all three characterization methods and for both enhancements, 2) The characteristics of the reduced input sets are not reference-like, and 3) SimPoint and SMARTS, the two sampling techniques, are extremely accurate and have the best speed versus accuracy trade-offs. Finally, this paper presents a decision tree which can help architects choose the most appropriate technique for their simulations.*

1. Introduction

The SPEC 2000 benchmark suite [11] is the current *de facto* standard for simulation-based computer architecture research. The largest input set for each benchmark in this suite is called the `reference` input set. Although this input set typically yields the most realistic behavior, it is rarely simulated to completion due to its very long simulation time.

Since these lengthy simulation times preclude a detailed exploration of the design space, computer architects resort to alternative simulation techniques to reduce the simulation time. These techniques include: reducing the size of the input set, simulating a piece of the program that is presumed to be representative of 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.

To address this issue, this paper evaluates the accuracy of the six most prevalent techniques – with respect to the `reference` input set – by characterizing them using three different methods. The six techniques are: 1) SimPoint [18], 2) Reduced input sets (MinneSPEC [13] and SPEC `test` and `train`), 3) Simulating the first `Z` million instructions only, 4) Fast-forwarding `X` million instructions and then simulating the next `Z` million, 5) Fast-forwarding `X` million, warming-up the processor for the next `Y` million, and then simulating the next `Z` million instructions, and 6) SMARTS, which is a rigorous, statistically-based sampling technique [20]. The three methods used to characterize these techniques are the: A) Processor bottleneck, B) Execution profile, and C) Architectural Level characterizations. After determining the accuracy of these techniques using these characterizations, we then analyze each from three additional perspectives, namely: their speed versus accuracy trade-off, potential configuration dependence, and the fidelity of their performance bottlenecks. Finally, this paper quantifies the effect that these techniques can have on the execution time for two microarchitectural enhancements – *simplifying and eliminating trivial computations* [22], and *next-line prefetching* [12]. The purpose of these comparisons is to determine the effect that each technique's inaccuracies could have on the apparent speedup of an enhancement, as compared to the actual speedup when using the `reference` input set.

The contributions of this paper are as follows:

1. It characterizes the accuracy of the six most popular techniques, with respect to the `reference` input set, at the hardware, software, and architectural levels.

2. It compares the speed versus accuracy trade-off of each technique.
3. It examines the potential configuration dependence of each technique.
4. It shows how the induced error of each technique can affect the apparent performance improvement of two enhancements.
5. It presents a decision tree to help architects choose the technique that is best for their simulations.

The remainder of this paper is organized as follows: Section 2 describes the problem, in addition to describing each of the six techniques. Descriptions of the experimental framework and each characterization method are given in Sections 3 and 4, respectively, while the simulation results are given in Sections 5, 6, and 7. Section 8 describes some related work. Section 9 makes specific recommendations about simulation methodology and Section 10 concludes.

2. Prevailing Simulation Techniques

To reduce the simulation time to a tractable level, several techniques are commonly used 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` inputs. In addition, developing reduced input sets can be a very tedious and time-consuming process. Examples of 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 benchmark is simulated for a fixed number of instructions while presuming that that arbitrary sample 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 (*i.e.* **Fast-Forward `X` + Run `Z`** or **FF `X` + Run `Z`**). This technique potentially improves on Run `Z` by skipping over the less interesting aspects of the program. One problem with FF `X` + Run `Z` is that, after fast-forwarding, the processor and memory states are "cold" (*i.e.* 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 [14]. The key to good results with population sampling is to ensure that the subset chosen accurately reflects the overall population. Three primary sampling techniques have been proposed for use in computer architecture research studies – *representative*, *periodic*, and *random* sampling.

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** [18] technique, for example, a relatively small number of simulation points are chosen to be 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 statistically-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 determines the overall simulation time.

By contrast, *periodic sampling* simulates selected portions of the dynamic instruction execution at fixed intervals. The sampling frequency and the length of each sample are used to control the overall simulation time; **SMARTS** (Sampling Microarchitectural Simulation) [20] is a recent example. To improve its accuracy, SMARTS uses 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 branch predictor and cache state.

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*. [6] suggested increasing the number of instructions dedicated to processor warm-up before each sample and/or increasing the number of samples.

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 last ten years of HPCA, ISCA, and MICRO to determine the most prevalent techniques. Our results show that the four most popular techniques are: *FF `X` + Run `Z`* (27.3% of all known techniques), *Run `Z`* (23.1%), *Reduced input sets* (18.5%), and simulating the benchmark to completion (17.8%). Since these four techniques account for almost 90% of all known

Table 1. The Final Specifics of the Candidate Simulation Techniques (Note: X+Y Mod 100M = 0)

Number of Permutations	Technique	Permutations
3	**SimPoint (Standard)**	Single 100M, Multiple 10M (max_K: 100) and 100M (max_K: 10) SimPoint 1.0, 7 Random Seeds (seedproj = 1), 100 iterations Warm-Up: Assume cache hit; 1M for 10M, 0M for 100M [10]
9	**SMARTS**	Detailed Simulation Length per Sample (U): 100, 1000, 10000 Warm-Up Length per Sample (W): 200, 2000, 20000 Initial Number of Samples (n): 10,000 Configuration: 99.7% Confidence Level, ±3% Confidence Interval [20]]
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

techniques, we included these four techniques in the set of candidate techniques studied in this paper. By contrast, we excluded random sampling since it was rarely used, despite it being a fairly well-known technique. We also included SimPoint and SMARTS in our final set since they are likely in increase in frequency. Finally, we included FF `X` + WU `Y` + Run `Z`, since it is a more accurate version of FF `X` + Run `Z`. Table 1 shows our final list of the 69 permutations of the candidate techniques. The values of `X`, `Y`, and `Z` were based on the superset of common permutations that we found in our survey. The specific values for SimPoint and SMARTS were based on those from [4, 10, 18, 20, 21].

3. Experimental Framework

In this paper, we used `wattch` [3] as the base simulator. 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 its 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 [5]. The total simulation time limited the number of benchmarks that we could simulate. Even then, to simulate the `reference` input set and the 69 permutations in Table 1 for 56 configurations and 10 benchmarks required the simulation of over ***1 quadrillion*** (10^{15}) detailed instructions, which required approximately ***40 CPU-years***. All benchmarks were compiled at optimization level O3 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 when more than one was available.

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.

4.1. Processor Bottleneck Characterization

The first characterization method is a performance bottleneck analysis using a Plackett and Burman design [17], or PB design. For architects, the PB design can determine which processor and memory parameters have the largest effect on the performance, *i.e.* are the biggest performance bottlenecks. The output of a PB design is a value that is associated with each input parameter. The magnitude of this number represents the effect that that parameter has on the variability in the output value, *e.g.* 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 sub-system.

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 the two rank vectors. Therefore, the technique that has the smallest Euclidean distance is the one that is the most accurate, *i.e.* has the set of performance bottlenecks that is most similar to those of the `reference` input set. (It is important to note

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

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

that we verified that using ranks did not significantly distort the results, as compared to using the PB magnitudes. Rather, using ranks prevented single parameters from dominating the results, which allowed less significant parameters to have some, limited, effect.)

Finally, our set of processor and memory parameter values is similar to those found in [22].

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 is executed, we increment that basic block's counter by the number of instructions that were executed in that instance of that basic block. This characterization factors in the number of instructions in each basic block.

We use a χ^2 test [15] 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. Architecture Level Characterization

The last characterization method that we used to compare techniques is at the architectural level. We first vectorize a set of metrics (IPC, branch prediction accuracy, L1 D-Cache hit rate, and L2 Cache hit rate), after normalizing each metric 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 architecture level characterization. These parameter values were chosen based on a survey of several commercial processors.

5. Results of Characterization Methods

The next three sections present the results of our analysis of the accuracy and simulation speed for the six techniques specified in Section 2. Section 5 presents the results for the three characterization methods while Section 6 describes the speed versus accuracy trade-off of each and the potential

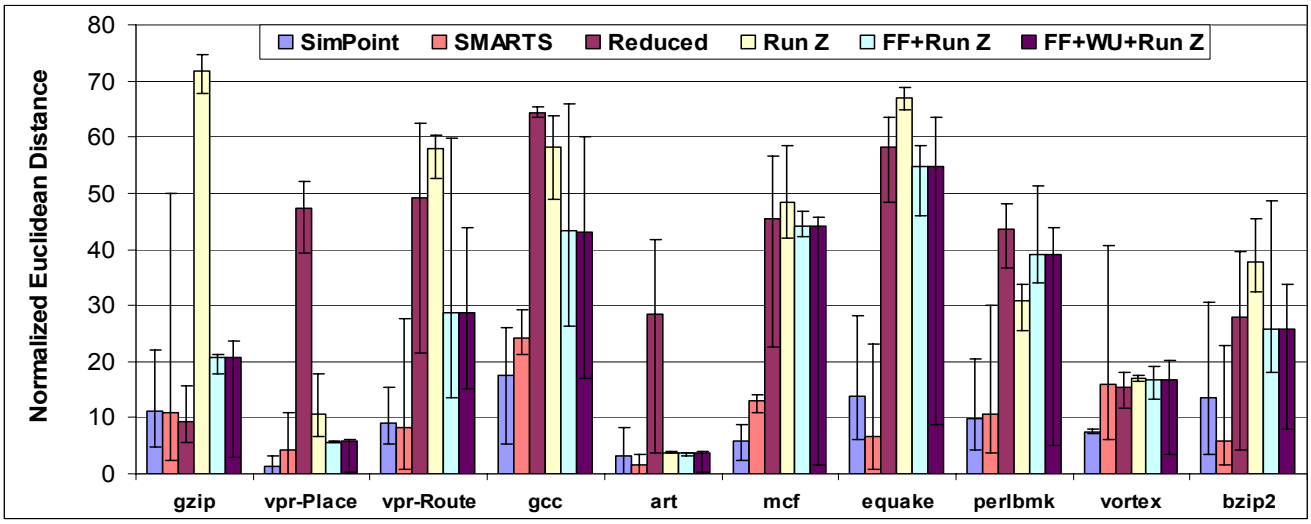

Figure 1. The Normalized Euclidean Distance Away from the reference Input Set for Each Type of Simulation Technique for the Performance Bottleneck Characterization. For Each Technique, the Average (Mean) Distance is Shown Along with the Minimum and Maximum Distance (Error Bars).

configuration dependence that each of these techniques could have. Finally, in Section 7, we illustrate how the inaccuracies of each technique can affect the apparent speedup results when evaluating a processor enhancement and a memory hierarchy enhancement.

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", *i.e.* <43, 42, 41, … 3, 2, 1> versus <1, 2, 3, … 41, 42, 43>); this distance is 162.75. Figure 1 presents the average distance for each type of technique, 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 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 parameters. For example, in *gcc*, the rank of the memory latency for the `reference` input set is 3 while its rank for the SPEC `test` reduced input set is 41. Second, our results for the basic block analysis, presented in Section 5.2, show 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 (*i.e.* Run `Z` variants) 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 is the same. First, since the values of `X`, `Y`, and `Z` are chosen arbitrarily, these three techniques simulate a portion of the program that not only may be uninteresting, but that may also be unrepresentative 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.

In general, there is a very small distance between SimPoint or SMARTS and the `reference` input set, in terms of their performance bottlenecks. At the very least, the distances for these two techniques are much lower than that of the other techniques. Overall, SMARTS is slightly more accurate than SimPoint since, for 6 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 5 benchmarks.)

It is important to note that large differences in the ranks for parameters that are not significant can increase the apparent distance for a technique. To examine if this is the case, Figure 2 shows the difference in the SimPoint and SMARTS distances, with respect to the `reference` input set, *i.e.* ||SimPoint – `reference`|| – ||SMARTS – `reference`||. Figure 2 only shows the results for the most accurate (smallest Euclidean distance) permutation.

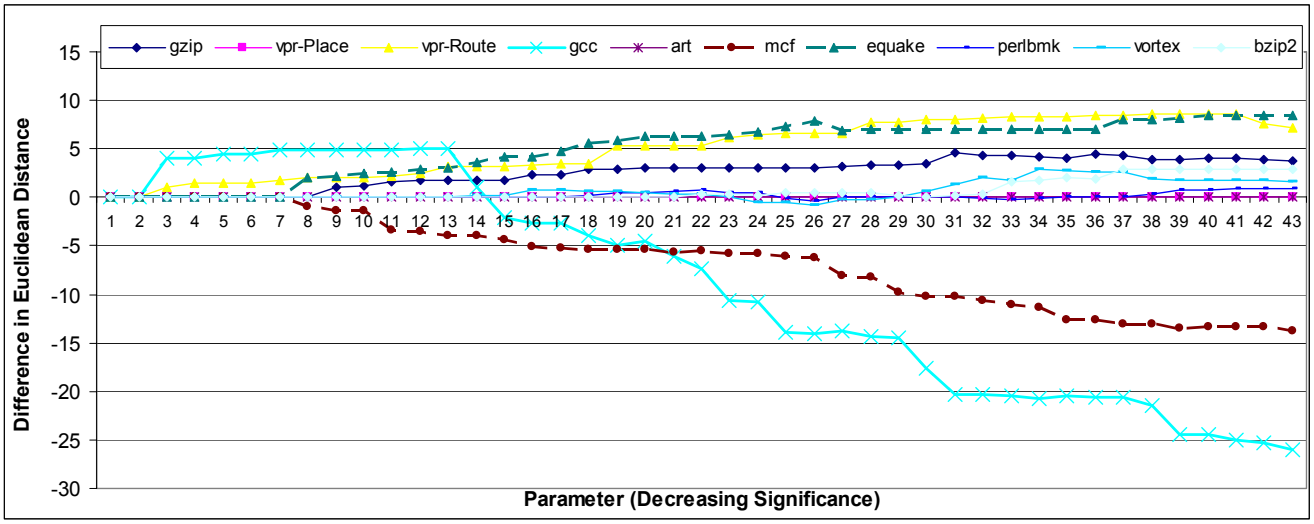

Figure 2. Difference in the SimPoint and SMARTS Euclidean Distances in Ascending Order of Rank

The parameters along the X-axis are sorted in ascending order of `reference` input set rank for *each benchmark*. Therefore, the same index in different benchmarks may correspond to different parameters. This plotting allows us to examine the effect of each parameter in decreasing order of significance in each benchmark. The difference in the distances for parameter N is the difference in the distances when only the N most significant parameters are included in the distance calculations.

For all benchmarks except for *gcc*, there is very little difference between the Euclidean distances for SimPoint and SMARTS, at least for the most significant parameters. 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 parameter 3 (memory latency) because SimPoint underestimates the significance of the memory latency for this benchmark. This is due to the fact that *gcc* has very complex phase behavior and that for this specific SimPoint configuration (multiple 10M simulation points), phase transitions are typically not chosen to be simulation points [4], which subsequently underestimates the effect of the memory latency. However, increasing the maximum number of simulation points, *e.g.* using 1M simulation points with a max_K of 300, can minimize this problem.

In conclusion, the results in this section show that the reduced input set and truncated execution techniques are very inaccurate compared to the results obtained by the `reference` input set. By contrast, SimPoint and SMARTS are both very accurate techniques, although SimPoint slightly underestimates the effect of the memory latency in *gcc*.

5.2. Execution Profile and Architectural Level Characterization Results and Analysis

In this section, we examine how the techniques compare to the `reference` input set when using the execution profile and architectural level characterizations. Since the results from both of these characterizations are fully coherent with those presented in the previous section, and due to space limitations, we omit tables of these results. Furthermore, since the results of the BBEF and BBV are virtually identical, we discuss only the results of the BBV characterization.

For the execution profile characterization, the results show that almost all permutations for all techniques executed a statistically similar set of basic blocks as the `reference` input set executed. However, the reason that the execution profiles for almost all permutations were statistically similar is because there were an extremely large number of basic blocks for the `reference` input set which results in a very large χ^2 statistic. That being stated, the results from this characterization show that the reduced input set and truncated execution techniques still have very different execution profiles than the `reference` input set. This result is not surprising since truncated execution simulates a small portion of the program and since reduced input sets do not simulate the same pieces of the benchmark at the same frequencies as the `reference` input set. On the other hand, the execution profiles for SMARTS and SimPoint are very similar to that of the `reference` input set, although SMARTS is more similar.

The conclusions from the results of the architectural level characterization are the same as the conclusions from the performance bottleneck and the execution-profile characterizations. Namely, the reduced input set and truncated execution techniques yield very different architectural metrics than does the `reference` input set while the architectural metrics for SimPoint and SMARTS are much more similar. These results, of course, are not surprising given the results of the other two characterizations.

It is extremely important to note that since these three characterizations examine the accuracy of the six techniques

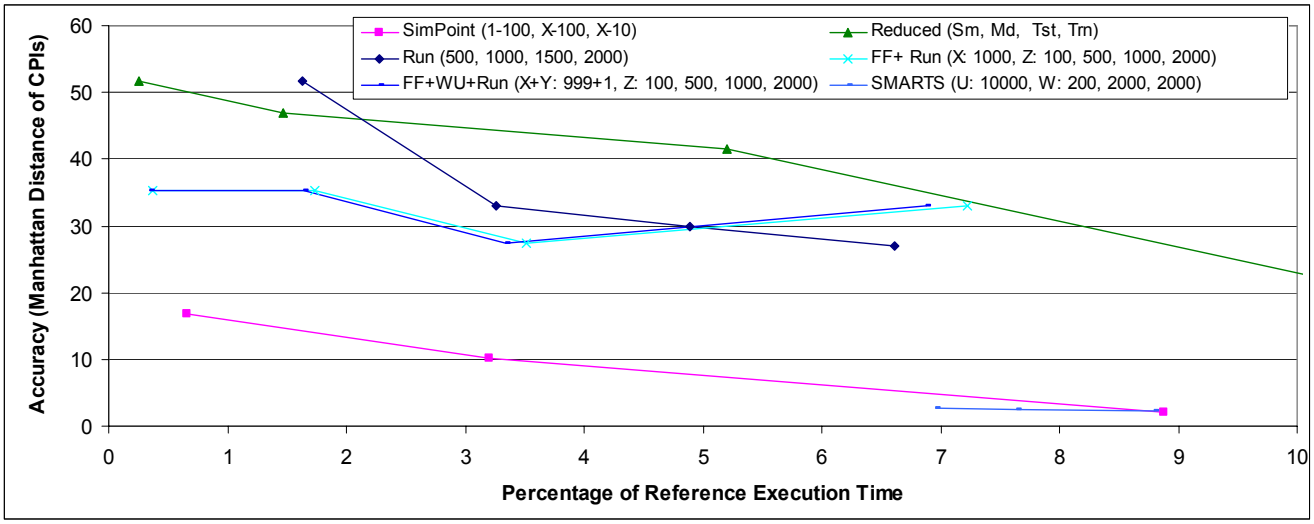

Figure 3. Simulation Speed versus Accuracy Trade-Off Graph of *gcc*

from different perspectives, the coherency of the results indicates that the accuracy of each technique is not merely a coincidental averaging of inaccuracies, but rather an intrinsic property of the technique. Therefore, although the conclusions are the same for all three characterizations, this coherency across all three bolsters the validity of the conclusions and the efficacy of the characterizations.

6. An Analysis of Speed versus Accuracy Trade-Off and Potential Configuration Dependence

6.1 Speed versus Accuracy Trade-Off Analysis

It is typically assumed 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 pre-eminent characteristic, speed emerges as an important consideration when the accuracies of several techniques are similar.

To accurately determine the speed versus accuracy trade-off (SvAT) of these techniques, we stipulated that all simulations run on the same machine to eliminate differences in the processor, memory sub-system, network, operating system, *etc.* Due to the number of simulations (approximately 30,000 for this analysis), all test cases were simulated only once. Therefore, measurement error could very slightly alter the results. In total, approximately 50 configurations, which represent the envelope of the hypercube of potential configurations, were simulated for each technique.

Figures 3 and 4 present the SvAT graphs of *gcc* and *mcf*, respectively, which were fairly representative of all benchmarks and were the most interesting. 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 `reference` input set's total simulation time while 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 since it more clearly presented the results.) We included the cost of generating simulation points (for SimPoint) and simulation checkpoints (for SimPoint and the truncated execution techniques) into the simulation speed. (Note: SimPoint 2.0 [19] dramatically reduces the time needed to determine the simulation points; it was not available when we started this study. However, for *gcc* and *mcf*, the cost of generating simulation checkpoints is the dominant non-simulation cost.) The cost of generating the reduced input sets and the initial profiling of SMARTS was not included as these costs were not quantified in [13] and [20], respectively. However, for SMARTS, the simulation times of the simulations that did not sample at a high enough frequency, *i.e.* required additional simulations, were included in the cost. (Across all benchmarks, the average number of simulations for each SMARTS permutation ranged from 1 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 [21].) Including or excluding the cost of a technique merely moves/stretches the technique's lines right or left, respectively. Finally, only the highest accuracy permutation of each technique is presented for the FF `X` + Run `Z`, FF `X` + WU `Y` + Run `Z`, and SMARTS techniques. The legend specifies the exact permutation.

The first key conclusion that we draw from these two figures is that the SvAT of reduced input set and the truncated execution techniques is very poor. Not only is their accuracy very poor, their poor accuracy is compounded by long simulation times. In particular, the `train` input set has the worst SvAT since its accuracy ranks towards the bottom and since its simulation time is significantly longer than any other technique. Therefore, the reduced input set and truncated

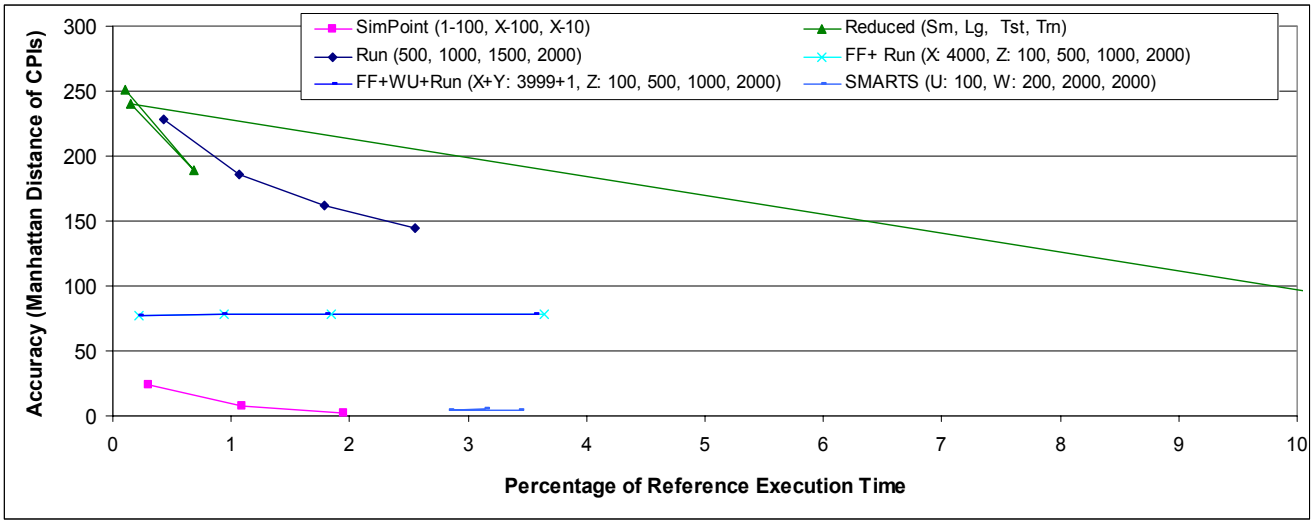

Figure 4. Simulation Speed versus Accuracy Trade-Off Graph of *mcf*

execution techniques, from the viewpoints of simulation accuracy and speed, do not offer any advantages compared to SimPoint and SMARTS.

It is interesting to note that due to *gcc*'s highly complex phase behavior, increasing the detailed simulation period of the truncated execution techniques does not automatically confer higher accuracy. Rather, blithely increasing the simulation period can simultaneously decrease both the simulation accuracy and speed.

The second key conclusion shown in Figures 3 and 4 is that, on average, SMARTS is the most accurate technique, which was one of the key conclusions of Section 5. (The accuracy of all nine SMARTS permutations was very similar.) Although the accuracy of SimPoint is not quite as good as SMARTS, SimPoint has a better SvAT than does SMARTS, 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 [16]). 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 increased simulation speed (and who isn't around deadline time?), then SimPoint is the most appropriate technique.

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

6.2 Potential Configuration Dependence

Another relevant consideration for architects is how the accuracy of these techniques changes based on the processor configuration. The 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. There is a configuration dependence when there are a large number of configurations in the higher CPI error ranges and/or if error does not trend. Figure 5 shows the percentage of configurations that fell into each range of CPI errors for that specific permutation across all benchmarks. For each technique, two permutations, worst (left) and best (right), are shown. A permutation was selected as the worst or best when it had the lowest or highest percentage, respectively, of configurations in the 0% to 3% error range.

Figure 5 shows three key results. First, since the CPI accuracy of the reduced input set and truncated execution techniques is extremely poor, both of these types of techniques have a significant configuration dependence. The second key result is that SMARTS has virtually no configuration dependence. Even for the worst permutation, in almost 80% of all configurations, SMARTS still yields a CPI value that is within 3% of the actual `reference` input set CPI. In the best permutation, this percentage climbs to almost 98%. However, this percentage is slightly less than the target of 99.7% of the configurations being within ±3% of the `reference` input set's CPI [20]. Nevertheless, given the "extreme" nature of the configurations – they represent configurations at the envelope of the hypercube of potential configurations – we conclude that SMARTS has virtually no configuration dependence. Finally, 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% is higher than the

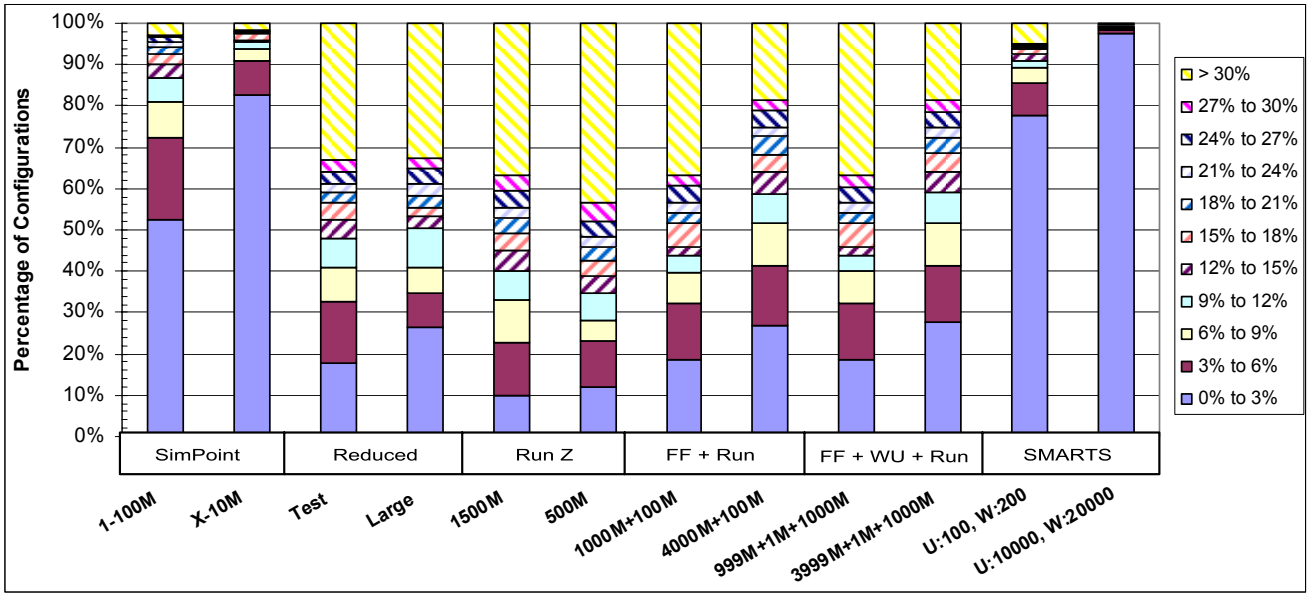

Figure 5. Configuration Dependence: Histogram of CPI Error (Relative to `reference`) for All Benchmarks

best case for SMARTS.

Although the results in Figure 5 show the frequency of CPI errors across all benchmarks, we found that the results presented in Figure 5 were fairly typical for each benchmark and that no benchmarks were "outliers" in terms of their frequency of CPI error.

Although the absolute accuracies may not be perfect, another key question regarding configuration dependence is: "Is the relative accuracy constant?" For the reduced input set and truncated execution techniques, the relative accuracy is not constant, in that the CPI error for these two types of techniques is not consistently positive or negative. Rather, as is especially prominent in *gcc*, for the same permutation, there are significant numbers of configurations that have CPI errors that are both less than -27% and greater than 27%. Therefore, for these techniques, the CPI error does not trend. For SimPoint and SMARTS, the relative accuracy is quite good, as least for the most accurate permutation of each technique. Even for *gcc*, the CPI error is consistently positive or negative. This conclusion confirms the results for SimPoint presented in [16].

In conclusion, the results in this section show that the reduced input set and truncated execution techniques have severe configuration dependences 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. From a pure CPI error point-of-view, however, the accuracy of SMARTS is the best.

7. The Impact of the Simulation Technique on the Evaluation of Enhancements

The results in Section 5 quantify the accuracy of each technique relative to the `reference` input set. What is not readily apparent, however, is whether there is a correlation between those differences and the effect a specific technique has on the simulation results. What is more important is whether these different techniques would result in different conclusions when evaluating a new architectural feature. To empirically determine the correlation between the accuracy of a technique and the observed simulation results produced when evaluating an enhancement, we quantify the induced error due to each technique for two microarchitectural enhancements, *Simplifying and Eliminating Trivial Computations* (TC) [22] and *Next-Line Prefetching* (NLP) [12]. We chose these two enhancements since TC targets the processor core and NLP targets the memory hierarchy. Furthermore, TC is a non-speculative enhancement while NLP is speculative.

The results, which are omitted due to space limitations (a detailed presentation of these results can be found in [24]), show that there is a distinct correlation between the accuracy of a technique and the accuracy of its apparent speedup results. Therefore, the techniques that are very accurate, such as SimPoint and SMARTS, have apparent speedups that are very close to the apparent speedup when using the `reference` input set. On the other hand, the inaccurate techniques – reduced input sets and truncated execution – produce apparent speedups that can be very different than the apparent speedup of the `reference` input set; in the worst case, the apparent speedups of these techniques overestimate and underestimate the apparent speedup of the `reference` input set by up to 100% (*e.g.* 0% speedup instead of 30%). What is especially troubling is that the speedups for these techniques are not consistently higher or lower than the speedup for the `reference` input set. This lack of consistency precludes the prospect of accounting for the

magnitude and the direction (positive or negative) of the error when examining speedup results. These differences are solely attributable to the fundamental inaccuracy of these techniques when they are compared to the `reference` input set.

8. 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. The papers that did evaluate 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. The most relevant related work falls into two categories: simulation methodology and simulator validation.

Simulation Methodology: Yi *et al.* [23] proposed using a PB design as a means of introducing statistical rigor into simulation methodology. More specifically, they used a PB design to identify the most significant parameters 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 while 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 analysis 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 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, that was present in an Alpha version of the SimpleScalar simulator. Their results showed that the simulators that model a generic machine (*i.e.* non-specific machine, 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 multi-processor 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%, which is higher than the speedups that are often reported for specific architectural enhancements.

9. Recommendations

Based on the results of the three characterization methods, the speed versus accuracy trade-off, and the configuration dependence analysis, we make the following 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 over the last ten years, and approximately one-third of the papers in recent 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, such as SimPoint and SMARTS, 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 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 and SMARTS – be used instead. While this may seem to be an intuitively obvious recommendation, the fraction of papers that used reduced input sets or truncated execution techniques actually increased from 68.9% in the eight years prior to the introduction of SimPoint, to 82.1% in the conferences that occurred after SimPoint was introduced. Finally, benchmarks from old benchmark suites should not be used unless there is 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 not to 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 three sections and from our experience in this study, Figure 6 presents the detailed ordering of the six 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 metrics), the speed versus accuracy trade-off, and the configuration dependence analysis.

The *Complexity to Use* category reflects the complexity of the changes that are needed to support that technique. Since the reduced input sets do not require any changes, they

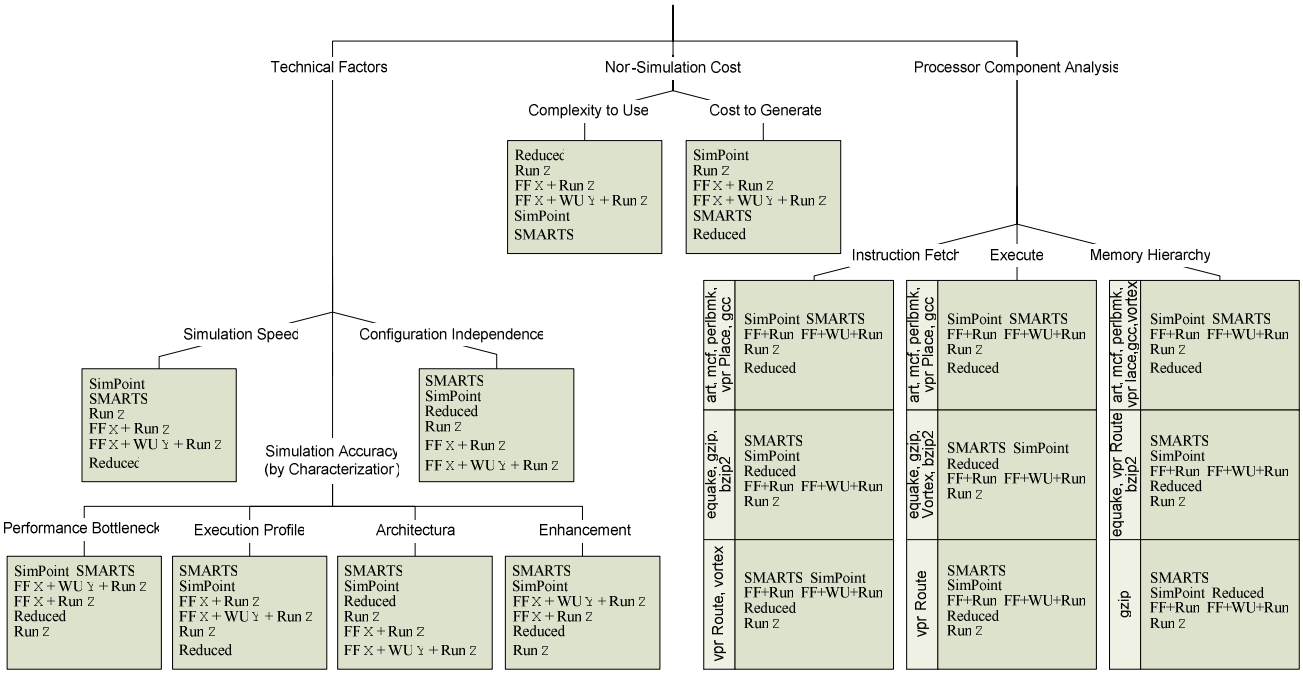

Figure 6. Decision Tree for the Selection of a Simulation Technique

have the lowest complexity to use. SMARTS has the highest complexity to use since it requires changes to the simulator to support periodic sampling, functional warming, and statistical calculations. The other four techniques have a medium complexity of use because they could require minor changes to the simulator to support fast-forwarding, warm-up, and early termination. The C*ost to Generate* category is the amount of effort that is needed "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 and reduced input sets have the highest costs to generate since new SMARTS parameters (U, W) may need to be found or new reduced input sets need to be created for each benchmark suite.

The *Processor Component Analysis* branch orders the techniques based on how similar their performance bottlenecks are within each of the processor's major components (*i.e.* Instruction Fetch, Execute, and Memory Hierarchy). After eliminating the parameters that are less significant than the dummy parameters (*i.e.* 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 parameters in the `reference` input set. By focusing on specific components, this analysis determines which techniques should be used for which components.

While 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 and SimPoint show the best behavior. For *gzip*, *equake* and *vpr-Route*, Run `Z` is not appropriate for any of the components and therefore should not be used. Reduced input sets are not appropriate when focusing on Instruction Fetch and Memory Hierarchy parameters for *gcc* and *vpr-Place*. On the other hand, for *gzip*, the reduced input sets are as accurate as SimPoint and SMARTS.

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

10. Conclusion

With the advent of popular execution-driven 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 alternative simulation techniques with the intent of decreasing the simulation time. Prevailing and emerging techniques fall into the categories of: 1) Reduced input sets, 2) Truncated execution, and 3) Sampling. In this paper, we characterized the accuracy of the MinneSPEC and SPEC reduced input sets; Run `Z`, FF `X` + Run `Z`, and FF `X` + WU `Y` + Run `Z` from the truncated execution category; and SimPoint and SMARTS from the sampling category.

We used three characterizations to determine the accuracy of each technique, with respect to the `reference` input set. First, we used the statistical Plackett and Burman

design to perform a performance bottleneck characterization of each technique. Second, we performed an execution profile analysis by tallying the basic block execution frequencies and instruction counts. Third, we used several architectural performance metrics as the final characterization method. After evaluating accuracy of these techniques with the previous three characterizations, we evaluated the speed versus accuracy trade-off and the potential configuration dependence of each technique. Finally, we then showed how the induced error of each technique can affect the performance of two microarchitectural enhancements.

To evaluate the accuracy, speed versus accuracy, configuration dependence, and the apparent effect on the speedup for the 69 permutations of the techniques for 10 benchmarks, we simulated over 10^{15} detailed instructions, which required about 40 CPU-years of simulation time.

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 two techniques is not offset by faster simulation speed, which further diminishes their utility. Second, these two techniques have significant configuration dependences because they have a high frequency of large CPI errors and because the CPI error does not trend. Third, as a result of these inaccuracies, the speedup results for these techniques does predictably trend.

Fourth, our results showed that, for all three characterizations and for the configuration dependence analysis, SimPoint and SMARTS are both very accurate techniques. Fifth, while SMARTS is slightly more accurate, SimPoint has a better speed versus accuracy trade-off. Finally, the most accurate permutation of both techniques produces very accurate estimates of the apparent speedup.

The final contribution of this paper is a decision tree to help architects choose the most appropriate technique(s) across on a wide range of categories.

Acknowledgments

We would like to especially thank B. Calder, R. Wunderlich, J. Lau, and M. Van Biesbrouck for answering our many SimPoint and SMARTS related questions. We would also like to thank B. Calder, Y. Chen, L. Eeckhout, C. Hescott, B. Kazar, J. Lin, K. Osowski, M. Tobin, H. Vandierendonck, K. Wu, and R. Wunderlich for their helpful comments on previous drafts of this work; B. Kochie and B. Runesha for their help and infinite patience in helping us finish our simulations; and to J. Lin, V. Packirisamy, K. Yellajyosula, and P. Yew for helping us set-up our simulations and allowing us to monopolize their computers.

This work was supported in part by National Science Foundation grants CCR-9900605 and EIA-9971666, the IBM Corporation, and the Minnesota Supercomputing Institute.

References

[1] B. Black and J. Shen, “Calibration of Microprocessor Performance Models”, IEEE Computer, Vol. 31, No. 5, May 1998, Pages 59-65.

[2] D. Burger and T. Austin, “The SimpleScalar Tool Set, Version 2.0”, University of Wisconsin-Madison Computer Sciences Department Technical Report #1342, 1997.

[3] D. Brooks *et al.*, “Wattch: A Framework for Architectural-Level Power Analysis and Optimizations”, International Symposium on Computer Architecture 2000.

[4] B. Calder *et al.*, Personal Communications.

[5] D. Citron, “MisSPECulation: Partial and Misleading Use of SPEC CPU2000 in Computer Architecture Conferences”, Panel Discussion in International Symposium on Computer Architecture 2003.

[6] T. Conte *et al.*, “Reducing State Loss for Effective Trace Sampling of Superscalar Processors”, International Conference on Computer Design, 1996.

[7] R. Desikan *et al.*, “Measuring Experimental Error in Microprocessor Simulation”, International Symposium on Computer Architecture, 2001.

[8] L. Eeckhout *et al.*, “Workload Design: Selecting Representative Program-Input Pairs”; International Conference on Parallel Architectures and Compilation Techniques, 2002.

[9] J. Gibson *et al.*, “FLASH vs. (Simulated) FLASH: Closing the Simulation Loop”, International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[10] G. Hamerly *et al.*, “How to Use SimPoint to Pick Simulation Points”, ACM SIGMETRIC Performance Evaluation Review, 2004.

[11] J. Henning, “SPEC CPU2000: Measuring CPU Performance in the New Millennium”, IEEE Computer, Vol. 33, No. 7, July 2000; Pages 28-35.

[12] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-associative Cache and Prefetch Buffers," International Symposium on Computer Architecture, 1990.

[13] A. KleinOsowski and D. Lilja, “MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research”, Vol. 1, June 2002.

[14] P. Levy and S. Lemeshow, “Sampling of Populations: Methods and Applications”, John Wiley and Sons, 1999.

[15] D. Lilja, “Measuring Computer Performance”, Cambridge University Press, 2000.

[16] E. Perelman *et al.*, “Picking Statistically Valid and Early Simulation Points”, International Conference on Parallel Architectures and Compilation Techniques, 2003

[17] R. Plackett and J. Burman, “The Design of Optimum Multifactorial Experiments”, Biometrika, Vol. 33, Issue 4, June 1946, Pages 305-325.

[18] T. Sherwood *et al.*, “Automatically Characterizing Large Scale Program Behavior”, International Conference on Architectural Support for Programming Languages and Operating Systems, 2002.

[19] http://www.cs.ucsd.edu/~calder/simpoint

[20] R. Wunderlich *et al.*, “SMARTS: Accelerating Microarchitectural Simulation via Rigorous Statistical Sampling”, International Symposium on Computer Architecture, 2003.

[21] R. Wunderlich, Personal Communications.

[22] J. Yi and D. Lilja, “Improving Processor Performance by Simplifying and Bypassing Trivial Computations”, International Conference on Computer Design, 2002.

[23] J. Yi *et al.*, “A Statistically-Rigorous Approach for Improving Simulation Methodology”, International Symposium on High-Performance Computer Architecture, 2003.

[24] J. Yi *et al.*, “Characterizing and Comparing Prevailing Simulation Techniques”, ARCTiC Technical Report 04-06, 2004.