

Phase-Guided Small-Sample Simulation

Joshua L. Kihm, Samuel D. Strom, and Daniel A. Connors

University of Colorado at Boulder
Department of Electrical and Computer Engineering
UCB 425, Boulder, CO, 80309
{kih, stroms, dconnors}@colorado.edu

Abstract

Detailed cycle-accurate simulation is a critical component of processor design. However, with the increasing complexity of modern processors and application workloads, full detailed simulation is prohibitively slow and thereby severely limits design space exploration. Sampled simulation techniques eliminate the need for full simulation by simulating in detail a very small but representative subset of a target application’s overall execution. Two effective and accurate sampling techniques are phase-based simulation and small-sample simulation. Both of these techniques have been adopted by the architecture design and simulation communities for research. However, both techniques were derived using a single benchmark evaluation suite and promote the same sampling method for all applications. Alternatively, an execution-aware sampling-based simulation technique can adapt during execution characteristics of the individual application being simulated and achieve the most efficient and accurate simulation acceleration.

To evaluate the impact of application characteristics on simulation approaches, we compare several simulation techniques using the *Spec2000* benchmark suite. Our results yield key conclusions about combining the strengths of previous simulation techniques into a single approach: (PGSS) Phase-Guided Small-Sample Simulation. PGSS adapts sampling to the characteristics of the application, thereby achieving high sampling accuracy and requiring an order of magnitude less detailed simulation time than previous techniques.

1. Introduction

Cycle-accurate architectural simulation is a vital tool in exploring potential designs of modern processors. However, as processor designs become more complex, modeling and simulation become progressively slower. Detailed architectural simulation remains several orders of magnitude slower than native execution despite gains in processor performance. This problem is further compounded by the emergence of the ever larger and more complex benchmarks, such as the *Spec2006* suite ([14]). A full, detailed, cycle-accurate simulation of a modern benchmark suite on even a small number of potential processor alternatives can require months of simulation time.

One approach to remedying reducing simulation time is to use sampled simulation, where only a very small, but representative, portion of a benchmark is simulated in detail. Results from these samples are then extrapolated to estimate performance or other design information for the entire execution of evaluated application. Two sampling methodologies, SMARTS ([16]) and SimPoint ([11]) are commonly used for exploring architecture designs that have gained wide acceptance in the simulation community. Figure 1 illustrates the overall operation of the SMARTS and SimPoints techniques in addition to *Phase-Guided, Small Sample Sim-*

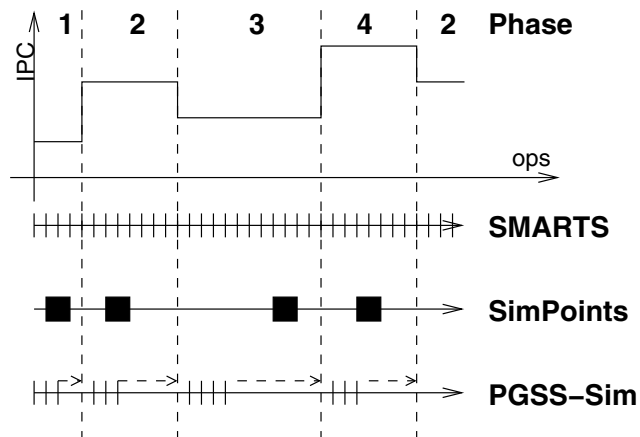


Figure 1. Illustration of SMARTS, SimPoints, and Phase-Guided, Small-Sample Simulation. SMARTS takes small, periodic samples regardless of phase. SimPoints takes a single, large sample in each phase. PGSS-Sim uses phase information to determine where small samples should be taken, thereby reducing the amount of detailed simulation.

ulation or PGSS-Sim technique proposed in this paper. As illustrated, SMARTS relies on very short periodic samples across the whole execution of a program. Conversely, SimPoints uses a longer representative sample of detailed simulation but only in each distinct application phase. PGSS-Sim combines these techniques by using phase information to guide when small samples are taken. This results in a technique which avoids the short-comings of both SMARTS and SimPoint while capitalizing on their strengths.

Although both SMARTS and SimPoints methods have been independently shown to be accurate and effective ([18]), both generate a single model used by all benchmarks being simulated. As both techniques were derived during the same time frame using the same benchmark suite, it is important to identify whether the simulation conclusions are specific to a set of applications. Previous work [19] demonstrated a similar concept of *drift* of characteristics that occurs between different generations of compilers and benchmarks. Thus the inherent characteristics of applications and their influence on simulation techniques is important to identifying robust simulation methods.

In this paper, the strengths of both approaches are combined into a technique called *Phase-Guided, Small-Sample Simulation* or *PGSS-Sim*. By tracking phase behavior during simulation, intelligent decisions can be made as to where to perform short periods of detailed simulation. Run-time phase change detection avoids the necessity of static binaries and limitations of clustering that limit SimPoints and takes into account the phase behaviors which

SMARTS ignores. Overall, the new PGSS phase detection properties greatly reduce the amount of detailed simulation needed over existing technique by over an order of magnitude.

The remainder of the paper is organized as follows. Section 3 explains in detail the PGSS-Sim technique and compares it to existing techniques. Section 4 goes into detail as to what constitutes a new phase change in PGSS. Section 5 compares the results from PGSS against previous sampling methods. Section 6 compares the simulation times of the tested techniques and, finally, Section 7 outlines conclusions and future work.

2. Background and Motivation

2.1 SimPoints: Simulation Sampling

The SimPoints system ([6, 4, 7, 10, 11]) exploits the repetitive nature of most programs to find representative samples of program execution. Most programs exhibit a small number of unique behaviors called *phases*. SimPoints works by finding the contribution of each phase to the overall program performance and a representative sample from each phase to be simulated in detail. Figure 1 demonstrates the SimPoints approach by showing a single interval of detailed simulation for each of the identified program phases 1,2,3, and 4.

SimPoints identifies phases by gathering performance data with respect to the execution of program code points through the *Basic Block Vector* or BBV. Frequency counters for each program basic block track the execution of the functional simulation of a program. A hardware implementation [8] was also proposed to collect BBV information for real hardware systems. SimPoints uses a clustering algorithm to group clusters of BBVs with similar sample behavior and partitions the benchmark execution into phases based on the clusters. The simulation sample closest to the center of the cluster is used to represent the entire phase and is simulated in detail. Estimating overall program performance is then simply a matter of calculating a weighted sum of the performance of each simulation point multiplied by the contribution of that phase.

The SimPoints system has two underlying limitations. First, SimPoints requires that the BBV be collected for the entire execution of a program. This requirement means that the BBV data collection and clustering analysis must be repeated for each version of a program as well as each input variation to the program. Second, the computational complexity and high memory demands of most clustering algorithms limits the number of BBVs that can be collected for a given binary. For example, for the iterative k-means clustering algorithm used by SimPoints, each sample must be compared to each cluster during each iteration, which scales linearly, but the number of iterations needed to reach a stable clustering also increases. A more indirect effect is that since each sample is compared to the clusters with each iteration, they all must be kept as high as possible in the memory hierarchy. A clustering of millions of samples will easily overrun even the largest of cache hierarchies. As such, phase behavior cannot be examined at a very fine granularity as it would produce an prohibitive number of BBVs to be clustered. Additionally, larger workloads, such as *Spec2006*, create millions of BBVs even at relatively coarse granularity. Thus for larger workloads the SimPoints clustering process is prohibitively slow.

Since SimPoints relies on clustering, there is a limitation in how small the samples can be. By using large samples, fine-grained behaviors can easily be missed by a SimPoints type analysis. Figure 2 illustrates the IPC versus completed operations of the *Spec2000* benchmark *164.zip* in a simulated environment. The IPC is measured over periods ranging from one hundred thousand to one hundred million operations. Although the benchmark shows periods of wild variations in IPC at very small measurement periods, this be-

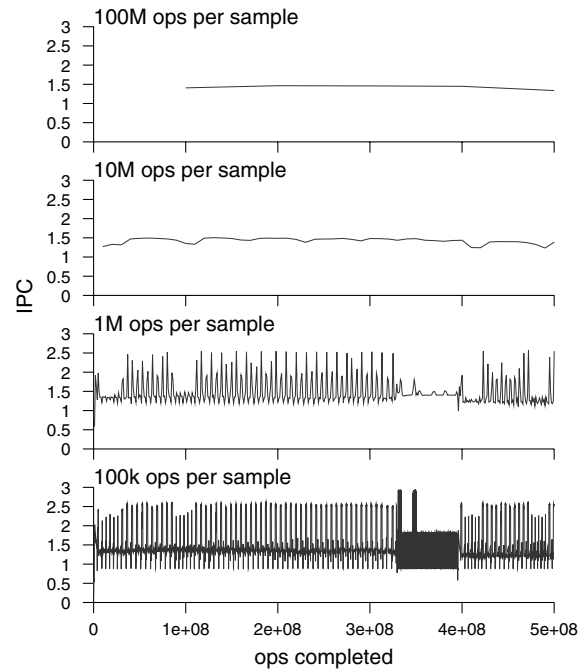


Figure 2. IPC versus Completed instructions for *164.zip* for different sampling periods. For clarity only the first 500 million instructions are shown.

havior is averaged out, and therefore invisible when the sampling period is large. The very methodology used specifically to identify and isolate program phases makes it impossible to identify recognizable fine-grained phases. Additionally, the detailed simulation of a phase must capture the invisible, finer-grained behaviors so that it is necessary to simulate a large number of instructions in each phase in order to gain an accurate picture of the behavior or behaviors of within that phase.

2.2 SMARTS: Small-Sample Simulation

Small-sample simulation was first proposed in [2] and was expanded in the SMARTS system ([16]). Very short, periodic samples of detailed simulation on the order of a thousand instructions are interleaved with longer periods, on the order of one million instructions, of functional simulation of the processor core. Figure 1 demonstrates the many small samples of SMARTS, uniformly occurring over the program simulation. Within SMARTS, a fast mode of simulation is used to keep the long-lifetime cache and branch-predictors warm. Additionally, each detailed simulation period is immediately preceded by an interval of three or four thousand instructions of detailed simulation in which statistics are not measured. This pre-sample simulation is used to warm up short-lifetime structures of the processor and greatly reduces the amount of overall detailed simulation needed. By taking a very large number of samples and looking at the variation among them, it is possible to statistically estimate the error induced by the sampling process. In fact, by using checkpoints and simulating samples in a random order, it is possible to estimate the overall performance within statistical confidence bounds by detailed simulation of only a small number of the samples ([15]).

The fundamental weakness of the SMARTS system is that it ignores program phase. SMARTS assumes that performance during the detailed simulation is representative of the entire fast forward-

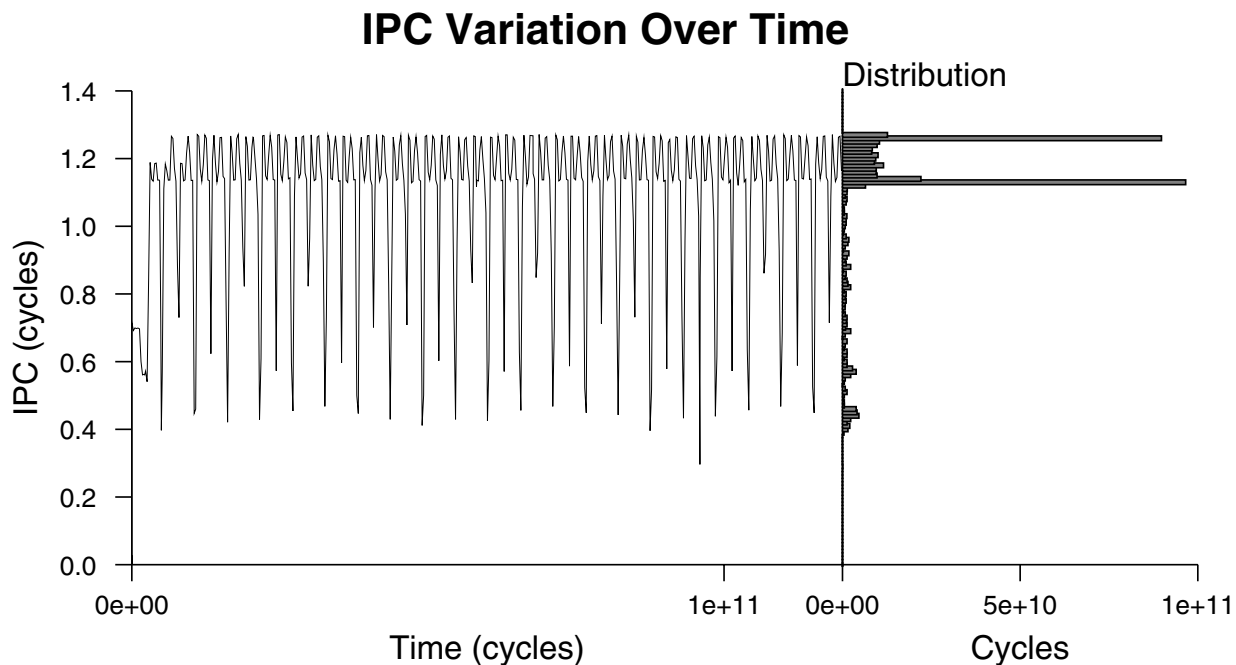


Figure 3. IPC versus time and distribution of IPC on a Pentium-4 execution of *168.wupwise*. For clarity, only the first 120 billion cycles of execution are shown, but the distribution is for the entire execution of approximately 420 billion cycles.

ing period. If the detailed simulation occurs immediately before a phase change, the sample will not be representative of a duration since it reflects a transition time. SMARTS works around this problem by taking samples very frequently and, as such, collects a very large number of samples. The statistical analysis used to determine the confidence intervals assumes that the population of samples is unimodal. That is, the samples follow a Gaussian distribution centered around a single value. As most programs exhibit phases, the distribution of samples typically follows a polymodal distribution around one value for each phase. Figure 3 illustrates this behavior through time-variant but repetitive nature of the *Spec2000* benchmark *168.wupwise*. Included with the time variation of IPC is The distribution of the approximate number of cycles spent in each IPC bin. The results clearly show a non-Gaussian distribution of IPC.

By assuming that the distribution is a single Gaussian, performance variation is overestimated. If a small, randomly-chosen set of the samples is used to estimate performance, a larger number of samples would be needed to meet confidence bounds because the variations between samples looks very large. Additionally, in this benchmark or any benchmark which shows wide variations in performance, phases that show very different performance may be over represented or not represented at all in a random sampling, leading to inaccurate results even if confidence bounds had been met. However, if phase behavior is considered, only a very small number of samples are needed from each phase to characterize that phase. The total number of samples needed across all phases is likely to be less than the number needed without phase distinction. It has been shown in [17] that by taking phase behavior into account in the SMARTS system, the number of samples needed can be reduced by over forty times over full SMARTS simulation. However, this work also points out the limitations of clustering that make it im-

possible to select proper samples at the fine granularity needed for SMARTS.

3. Methodology

Phase-Guided, Small-Sample Simulation brings together the strengths of the SMARTS ([16]) and the SimPoints ([11]) methodologies. SMARTS takes periodic, short samples without any regard to phase. SimPoints is based solely on phase data, but because of the need for offline data clustering, is limited to a very coarse granularity. By combining the techniques of SMARTS with those used in [12] for determining phase in order to guide run-time optimizations, it is possible to improve upon both techniques.

In [12], Sherwood et al. proposed hardware for tracking basic block execution frequencies in real time. This hardware tracks program phases at very large time scales as execution occurs. This technique avoids the troublesome limitations of clustering in SimPoints-style phase analysis and eliminates the need for offline analysis all together. Although the authors intended their ideas to be implemented in hardware and be used at a very coarse granularity on the order of one hundred million operations per sample, it is very easy to implement the algorithm in software and use it to track phase behaviors at a much finer granularity. We propose using the BBV information to guide a SMARTS style sampling method in PGSS-Sim.

Figure 4 illustrates the basic block frequency algorithm. Every time that a taken branch is encountered, the address of the branch and the number of retired operations since the last completed branch are recorded. The branch address is sent through a hash function which simply selects five bits from the address and concatenates them into an index for a register file. The five bits are chosen at random, but remain constant throughout the simulation. The entry at that index is then incremented by the number of

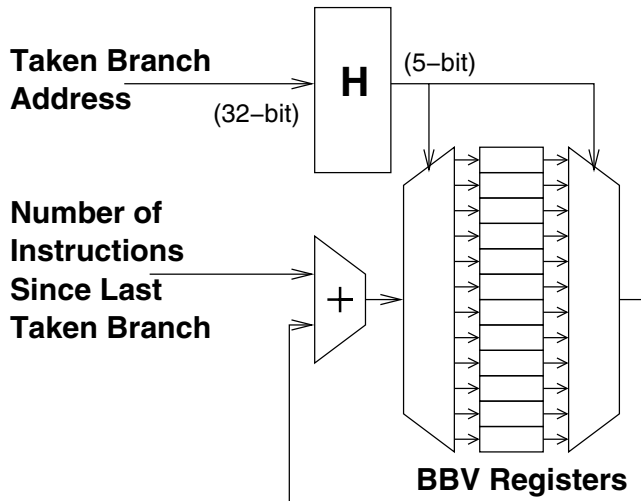


Figure 4. A hardware illustration of the algorithm used to track phase changes in Phase-Guided, Small-Sample Simulation. When a taken branch is encountered, five random bits from the address are taken from the branch’s address (the hash block) and used to address into a small register file. The register is incremented by the number of completed operations since the last taken branch.

retired operations. After each sampling period, the data in the registers are compiled into a BBV. The BBV is then compared against the data from previous samples to see if it has similar execution patterns to previous samples, that is, is it in the same phase. The data is normalized so that the vector has a L2-norm of one. The vectors can then be compared using a simple dot product which will yield the cosine of the angle between them. This avoids the problems of slightly different length samples and having to find absolute differences as in the Manhattan distances used in SimPoints. The threshold can be left in terms of the cosine of the angle as the angle cannot be more than $\frac{\pi}{2}$ and therefore, there is a one-to-one correlation between the angle and the cosine of the angle. Threshold determination is a very important aspect of phase detection as the threshold must be customized for each evaluated program. Section 4 explores the evaluation of phase threshold in the PGSS-Sim system.

Figure 5 illustrates the overall process of PGSS-Sim. The detailed warm-up and simulation occur just as they do in SMARTS. The only differences in the fast-forwarding step are that its length is shorter (one hundred thousand instructions rather than one million) and that the basic block frequencies are estimated as described above. The shorter fast forwarding periods allow for detection of very fine-grained phase information. The BBV is first compared to the last BBV since it is most likely that no phase change occurred. If the change in BBV is below some threshold, the performance and BBV data are added to the current phase’s profile. If not, the BBV is compared against all phases encountered to see if it falls within the threshold value of any previous phase. If it does not match a previous phase, a new phase is created. The next step is to see if the collected data from the phase falls within confidence bounds and detailed simulation of that phase can stop. Detailed simulation is skipped and another fast-forwarding period is started. If further data is needed on the phase, there is a check if the last detailed sample in this phase occurred within the last million instructions. This check aims to spread the samples across the occurrences of the phase in order to capture any temporal variation.

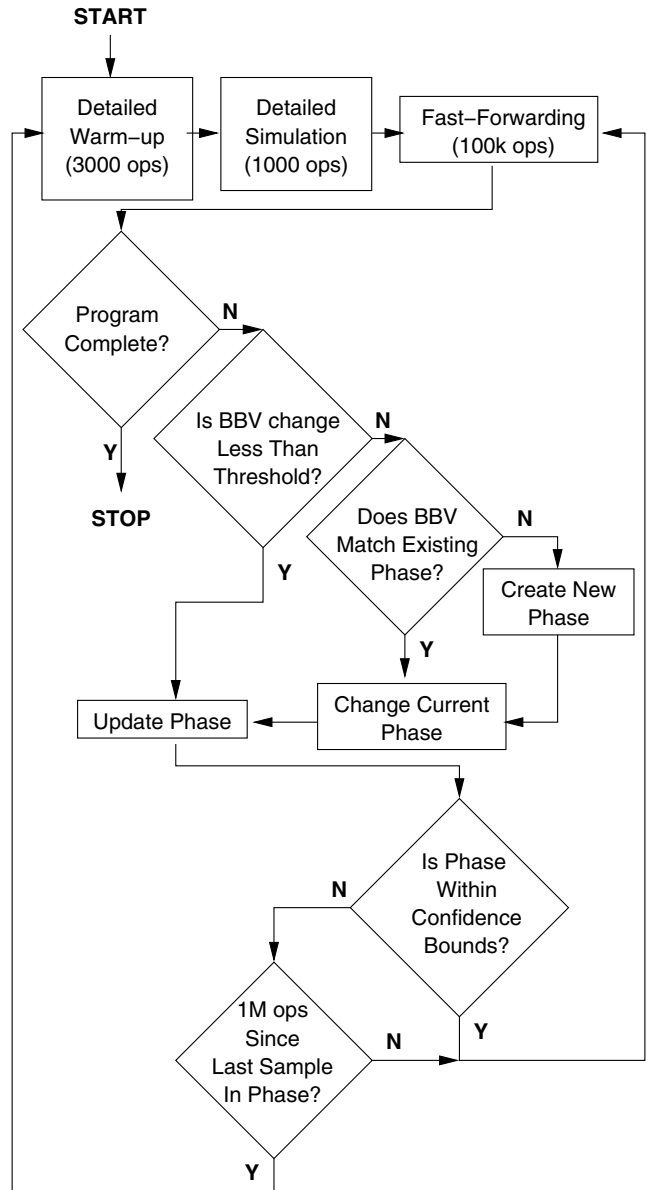


Figure 5. A flow chart of the PGSS-Sim technique. The detailed warm-up and simulation follow the SMARTS methodology. The one addition to the fast-forwarding technique is that basic-block information is recorded.

PGSS-Sim most closely resembles the work of Pereira et al. ([9]) in which BBV are tracked on-line for creating traces for embedded systems. The important difference is that phase behavior is tracked on a coarser granularity and only one, large sample is taken for each phase. Thus the phase change of the program must be predicted. Although simple models have been shown effective in predicting phase change, the process of detecting phases changes often requires detailed simulation of architecture components, thus requiring simulation overhead. Nevertheless, accurate phase detection is necessary to avoid long sampling periods in which phase intervals may have passed by the time they are detected. Further, because detailed trace generation is done for a large period of millions of instructions, phase detection can incur large overhead if per-

formed each time a sample is taken. If a new phase is predicted but does not occur, overhead is incurred without reward. Since PGSS-Sim tracks BBV behavior at a very small granularity, it is possible to respond to phase changes very quickly.

The operation of PGSS-Sim is illustrated in Figure 1 and offers several advantages over prior techniques. First, since only a very small amount of detailed execution are simulated at each sample, even if a sample is taken unnecessarily in a phase which has already been characterized, only a very small overhead is paid. Second, PGSS-Sim requires no complex phase prediction model. Additionally, the prior work assumes that the first occurrence of a given phase is representative of the phase as a whole. However, it is very possible that the first occurrence of a phase is subject to warming effects and therefore not be highly representative of the phase. In contrast, PGSS-Sim takes many samples throughout the occurrences of a phase to reduce this potential source of error. Finally, the prior work, much as the offline SimPoints methodology, uses the same amount of detailed simulation for each phase regardless of how often the phase occurs or how stable the phase's performance. PGSS-Sim automatically takes more samples in phases which occur a great deal or have a high amount of variance in performance and fewer samples in phases which are rarer or more stable. As such, PGSS-Sim outperforms prior work in terms of accuracy and requires significantly less detailed simulation as demonstrated in Section 5.

4. Determining a Phase Threshold

The most difficult and important consideration for the PGSS-Sim technique is where to set the BBV difference threshold. The threshold must be set so any change above the threshold corresponds to a significant change in performance and at the same time BBV differences below the threshold correspond to very similar performance. Having an absolute IPC change threshold makes comparisons between benchmarks impossible. An IPC change of .1 is fairly insignificant for a program with an IPC of 2.5, but is quite significant if the IPC is .5. Similarly, a percent change is difficult to characterize as it tends to over-emphasize low IPC periods. For instance, if a program goes through a period of very low IPC, it will likely show large percentage swings in IPC in percentage terms even if the performance is relatively stable. In order to compare data between benchmarks, all IPC changes are compared to the standard deviation of all samples across the benchmark illustrating the percentage of the total variation across the benchmark.

The changes between consecutive samples can be plotted on a simple two-dimensional graph where the X-axis corresponds to the change in BBV, expressed as an angle, and the Y-axis corresponds to the absolute change in IPC. The graph can be segmented first into two horizontal regions, one corresponding to IPC changes of interest and one to low IPC changes. Similarly, the graph can be segmented vertically into samples with BBV changes above and below the threshold. This is illustrated in Figure 6. The graph is divided into four regions. The goal is to decide on a BBV threshold that maximizes the number of samples which fall into the regions of detected phase changes (Region 2 in the figure) and small changes which are not above the threshold (Region 3 in the figure). At the same time, the number of samples which are undetected changes (Region 1) and false positives (Region 4) must be minimized.

Figure 7 shows the distribution of one-hundred thousand instruction samples for ten of the *Spec2000* benchmarks. The shade of each point indicates the percentage of samples which demonstrated that level of BBV change and that level of IPC change. For each benchmark, the IPC change is measured versus the standard deviation across all samples so that samples can be meaningfully compared against data from other benchmarks. All benchmarks are weighted equally so shorter benchmarks are equally weighted to

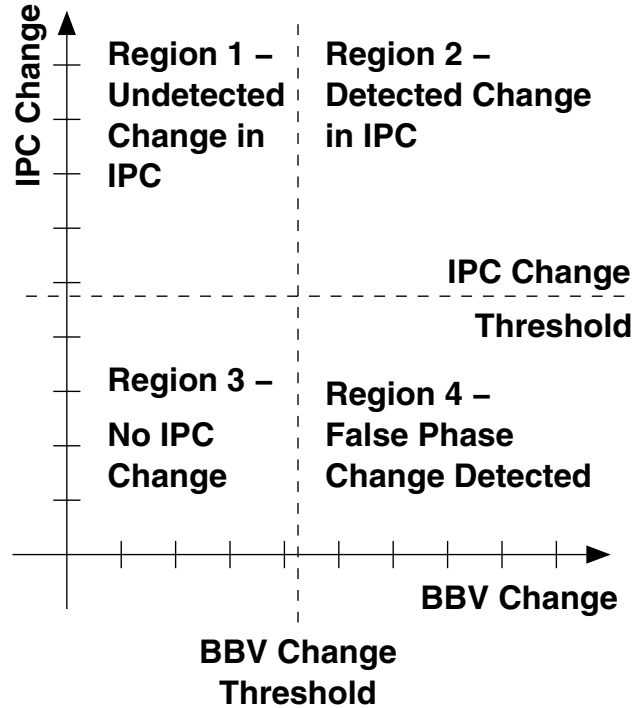


Figure 6. An illustration of how a given sample can be treated depending on what basic-block vector and IPC changes are considered to be significant.

longer benchmarks. It is evident from this graph that BBV changes greater than approximately $.05\pi$ radians typically correspond to a large change in IPC for the included benchmarks.

Another view of this data is shown in Figure 8. The graphs show what percentage of changes of various IPC levels are detected for various thresholds. In terms of Figures 6 and 7, this is the number of samples in Region 2 divided by the sum of the number of samples in regions 1 and 2. As expected, there is a knee in the curve around $.05\pi$ radians. Performance is better for larger IPC changes because small IPC changes often correspond to very small changes in BBV which may be very subtle phase changes or simply variations within a phase.

Figure 9 illustrates the percentage of detected phase changes which did not correspond to a large change in IPC. In Figure 6, this would be the number of samples in Region 4 divided by the number of samples in Region 2 and Region 4 combined. False positives are detrimental because they cause excess samples to be taken by creating a new phase where there is no difference in performance. False positives should be minimized by setting the threshold as high as possible, but not at the expense of missing important performance changes.

Although general conclusions about the effectiveness of different thresholds can be gained by looking at data in a larger context, the direct effects must also be investigated. The effectiveness of PGSS-Sim at reducing the amount of detailed simulation will be dependent on how many unique phases are detected, the stability of each phase, and how often phase transitions occur. Figure 10 illustrates these statistics for different thresholds for the *Spec2000* benchmark *300.twolf*. The standard deviation of all samples of *300.twolf* is very small (.055) and *300.twolf* has fairly weak phase behavior, especially at coarse granularity. However, at a fine granularity, *300.twolf* shows periodic, short periods of abnormally

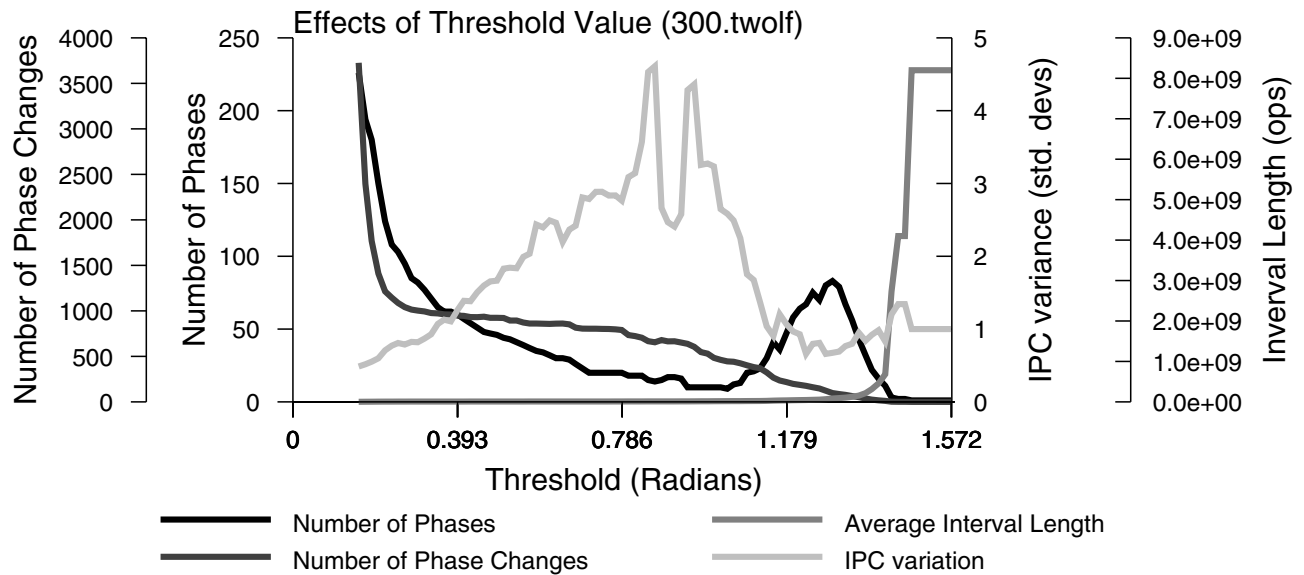


Figure 10. The effects of different thresholds on the measured phase characteristics of 300.twolf.

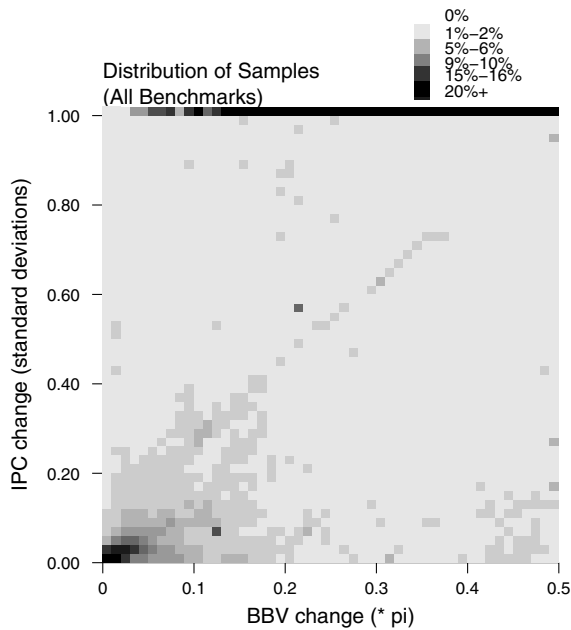


Figure 7. A two-dimensional distribution plot of IPC and basic-block vector changes between 100,000 operation samples across 10 Spec2000 Benchmarks.

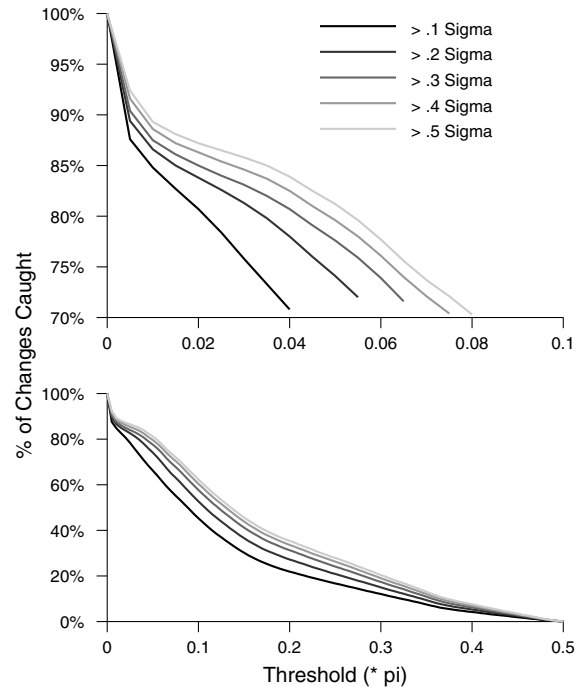


Figure 8. The percentage distribution of significant IPC changes detected for different basic-block vector changes.

high and low performance. The number of detected phases quickly drops as the threshold increases, but the variation in each phase raises quickly. Clearly the choice of threshold will have a large effect on the effectiveness of PGSS-Sim at both reducing the amount of detailed simulation and producing accurate results.

5. Results

Ten of the Spec2000 [13] benchmarks were tested with PGSS simulation with various parameter values. The first reference input was used for each benchmark. Additionally, the SimPoints ([3]), an on-line SimPoints variation ([9]), SMARTS ([16]), and TurboSmarts

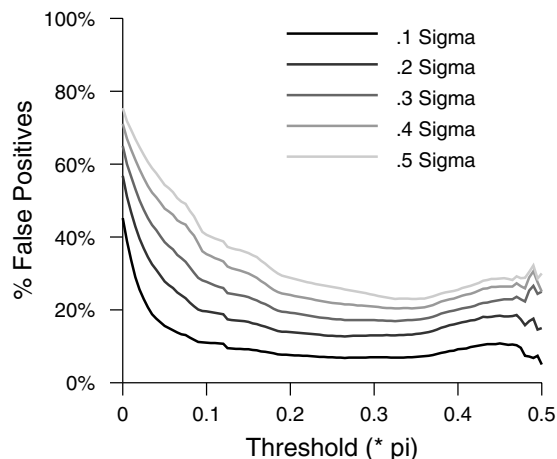


Figure 9. The percentage distribution of detected phase changes that are false positives for various threshold levels.

([15]) systems were tested for comparison. The simulation environment used is based on the IMPACT compiler tool chain ([1]). The processor simulated is a 4-wide issue, super-scaler, in order processor. This is attached to a two-level cache hierarchy consisting of a split first level (4-way associative, 64-kilobyte each for data and instruction) and a 1 megabyte unified Level2 cache. Although only one core was used in this experiment, this is meant to be roughly representative of a single core on a modern chip multiprocessor (CMP) system.

The performance of PGSS simulation is highly dependent on the values of certain parameters. The most important is the value of the BBV threshold, as discussed in Section 4, and the sampling period of the BBV. Figure 11 illustrates the sampling error for the ten tested benchmarks for various combinations of these parameters. The accuracy of PGSS-Sim varies widely between benchmarks and with changes in the parameters. Overall, PGSS-Sim needs long, stable phase intervals to be effective. This is dependent on both the nature of the program and how the phases are measured. When the BBV sampling is too short or the threshold value too low, the phase changes occurs too frequently and there are too many phases to accurately characterize, as many phases will only have a very small number of samples taken in them. If the threshold is too high, changes in performance may not be captured by the changing phase. If the sampling period is too long, then fine-grained phase behavior may become averaged out and mixed in with surrounding phases.

The difficult factor in setting these parameters is that they effect different benchmarks by unequal amounts. Although each benchmark performed best with a different set of parameters, using a sampling period of one million instructions coupled with a tight threshold of $.05\pi$ yielded the best overall results. Conversely, sampling of the benchmarks *179.art* and *181.mcf* performed very poorly at short BBV sampling periods. First, both benchmarks have very low IPC which inflated the error number when measured in percentage terms. Second, both exhibit high frequency variations in performance on the order of forty to fifty thousand operations. Since these behaviors are in no way synchronized with the BBV sampling, at high sampling frequencies, they tend to occur over two sampling periods. This means that many periods consist of two or three unique behaviors in different amounts. Even if another period encompasses the same micro-phases, it will likely contain

different amounts of them, as such, be characterized differently. More alarmingly, the detailed simulation taken at the beginning of the sampling period will not be indicative the period as a whole. On the other hand, if a larger period is used, a more coarse-grained picture is obtained where each period contains these small-scale behaviors at the same proportion as other periods and as the larger program interval. Although the small samples taken at the beginning are still not necessarily representative, as many of periods are grouped together into a phase, the constituent behaviors are represented in proportion and a more accurate picture of the phase is acquired.

The effectiveness of PGSS-Sim compares very well with other prevailing sampling techniques. The results are illustrated in Figure 12. The first technique tested is based on SMARTS ([16]). This is a very simple technique in that it relies only on periodic samples evenly spaced across the benchmark. Despite it's simplicity, it is very accurate, estimating the performance of most of the benchmarks tested within two percent accuracy. The trade off SMARTS makes however is that it uses a large number of samples. The TurboSMARTS ([15]) extension to SMARTS specifically addresses this issue by testing samples in a random order until they converge within certain statistical error bounds. The bounds used in this experiment were 3% accuracy with 99.7 confidence. However, this assumes a Gaussian distribution of samples, which is not the case with most programs. As such, the absolute error typically falls well outside these bounds, as it did in most of our experiments. In counting the amount of detailed simulation used for SMARTS, TurboSMARTS, and PGSS-Sim, the number of instructions executed in detailed warming and detailed simulation were counted. This was done because detailed warming is necessary before each sample and is just as slow as detailed simulation. This totals approximately four thousand instructions per sample.

The SimPoints ([3]) methodology was tested by performing an off-line clustering of the reduced BBV data from PGSS simulation. Different sampling periods of one, ten, and one hundred million instructions were tested and clusterings of five, ten, and twenty clusters were tested for each sample size. Additionally, thirty clusters of ten million instruction samples and three hundred clusters of one million instructions were tried for a total of eleven tests of each benchmark. Shown in Figure 12 are the sampling error for both the best clustering of each benchmark and the best-performing clustering overall (ten clusters and a sample size of one hundred million instructions). Since SimPoints details one period from each phase, the amount of detailed simulation is simply the sample size times the number of phases. Like SMARTS, SimPoints demonstrated extremely accurate sampling, but also required a very large amount of detailed simulation. An online version of SimPoints as outlined in [9] was also tested for various thresholds and sample sizes. For simplicity, a perfect phase predictor was simulated, that is, the phase profile was known prior to the actual simulation. Shown are both the overall best configuration (a sample size of one hundred million instructions and a threshold of $.1\pi$) and the average of the best performing configuration for each benchmark is shown.

For PGSS-Sim, all of the tests from Figure 11 were compared to the SMARTS and SimPoints results. Shown in Figure 12 are the best configuration for each benchmark as well as the best performing overall configuration (a sample size of one million instructions and a threshold of $.05\pi$). Although the accuracy numbers of PGSS are not as good as SMARTS or SimPoints, they are still very good and outperform TurboSMARTS. Additionally, PGSS-Sim achieves this accuracy with significantly less detailed simulation than TurboSMARTS, approximately an order of magnitude less than SMARTS, and two to three orders of magnitude less than SimPoints for both the online and offline phase analysis.

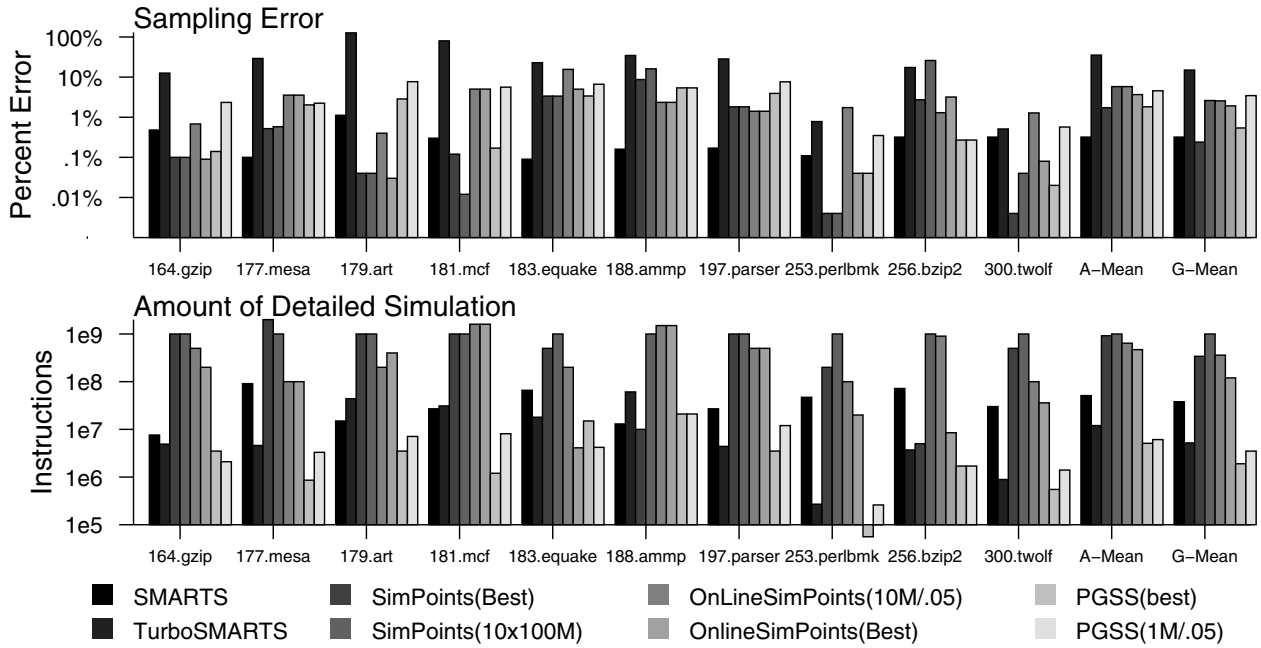


Figure 11. The sampling error as a percent of benchmark IPC for ten *Spec2000* benchmarks using three different BBV sampling periods and five different threshold values. Threshold values are given as a fraction of π .

6. Simulation Time

An important aspect of the performance of a sampling methodology is the reduction in simulation time. For PGSS-Sim, there are two factors which determine this performance. First, the overhead of the BBV tracking and second, the reduction in detailed simulation. As was seen in Figure 12, PGSS-Sim has at least an eight-fold reduction in detailed simulation over any of the other techniques tested. With regards to overhead, tracking of BBV data has a minimal impact on both detailed simulation, warming, and functional fast-forwarding. Functional fast-forwarding is non-cycle-accurate simulation which still keeps the cache hierarchy and branch predictors warm, which is necessary for SMARTS and PGSS-Sim. For functional fast-forwarding, no reduction in simulation time was observed. Because functional fast-forwarding tracks branch behavior in order to keep branch predictors warm, the added overhead of tracking them for BBV calculations is negligible. Similarly, the overhead for detailed warming and simulation was approximately one percent, which is well within the run-to-run noise in simulation rates. The exact simulation rates for the various levels of simulation the overall simulation times for the tested techniques are shown in Figure 13. In the figure, no checkpointing effects are taken into consideration. That is, it is assumed that no previous analysis of the benchmark has taken place. By using checkpointing in with a consistent binary, it would be possible to eliminate the need for fast forwarding in each of the techniques. Checkpoints for small-sample simulation can be made very small so that a very large number can be stored as was demonstrated in [15]. Although the overhead of loading checkpoints would still be present, overall the simulation time would be dominated by detailed simulation, where PGSS-Sim excels. In our simulator, PGSS-Sim requires only about three hundred, eighty seconds in total combined detailed warming and simulation time for the entirety of the ten benchmarks tested.

The reason why PGSS-Sim does not show very large simulation time advantages over the other techniques is the nature of our simulator. Fast-forwarding is only approximately four times faster than detailed simulation in our simulator. Two reasons exist for this. First, the simulator is execution driven on a non-native language meaning that our functional simulator is necessarily slow. Second, our detailed simulator is highly optimized and comparatively simple due to its RISC architecture and in-order design. In most simulators, there is a much wider disparity in performance between pure functional and detailed, cycle-accurate simulation meaning that reductions in detailed simulation will have a much larger effect on overall simulation time.

7. Conclusions and Future Work

By taking simulation-time phase information into account to guide small-sample simulation, it is possible to use a very small amount of detailed simulation to represent an entire program. PGSS-Sim requires the detailed simulation of an order of magnitude fewer instructions than SMARTS and two orders of magnitude less than SimPoints. Additionally, PGSS-Sim requires less detailed simulation and is more accurate than TurboSMARTS. These promising results open up several avenues of continued research. The *livepoints* used in [15] could easily be used to accelerate PGSS. Work is ongoing to extend PGSS to multithreaded and multicore processors. Since the optimal parameters for PGSS-Sim vary between benchmarks, these parameters must be automatically adjusted to each benchmark either in some sort of offline analysis of the benchmark or ideally, the algorithm would adapt at runtime to program characteristics. More accurately tracking exact phase transition points, as was proposed in [5], would both increase accuracy and reduce simulation time by more accurately capturing phase behavior. Finally, refinement of the technique would allow PGSS to match or at least approach the accuracy of SMARTS and SimPoints.

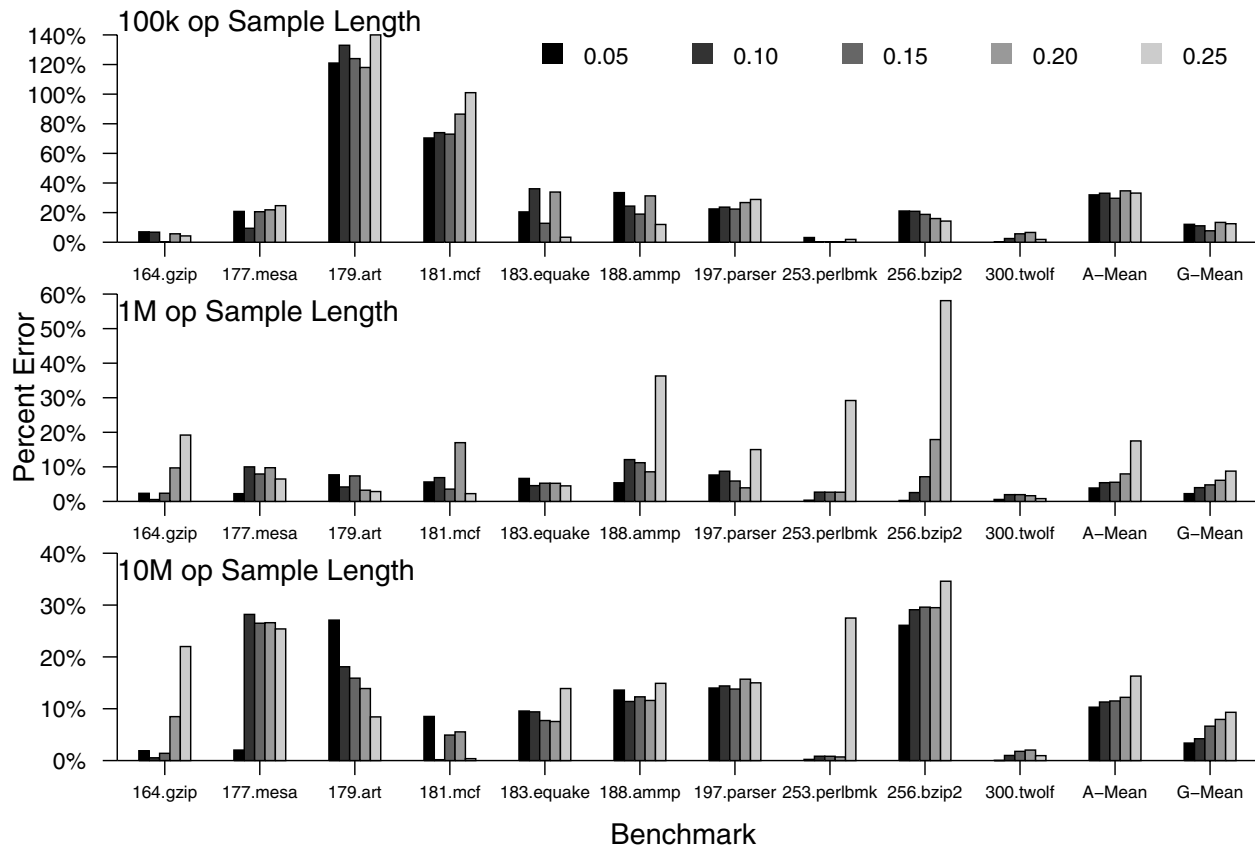


Figure 12. The sampling error as a percent of benchmark IPC and the amount of detailed simulation needed for ten *Spec2000* benchmarks using various sampling techniques. SimPoints and PGSS values are shown for the best configuration for each benchmark and the best overall configuration.

8. Acknowledgments

The authors would like to thank the anonymous reviewers and Paolo Faraboschi for their helpful comments and insights. Computer time was provided by NSF ARI Grant #CDA-9601817, NSF MRI Grant #CNS-0420873, NASA AIST grant #NAG2-1646, DOE SciDAC grant #DE-FG02-04ER63870, NSF sponsorship of the National Center for Atmospheric Research, and a grant from the IBM Shared University Research (SUR) program. The work was also sponsored by Intel Corporation and equipment grants from Apple Computer.

References

- [1] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu. IMPACT: An architectural framework for multiple-instruction-issue processors. In *Proceedings of the 18th International Symposium on Computer Architecture*, pages 266–275, May 1991.
- [2] T. Conte, M. A. Hirsch, and K. Menezes. Reducing state loss for effective sampling of superscalar processors. In *Proceedings of the 1996 International Conference on Computer Design (ICCD)*, October, 1996, 1996.
- [3] G. Hamerly, E. Perelman, J. Lau, and B. Calder. Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4), 2005.
- [4] J. Lau, E. P. G. Hamerly, T. Sherwood, and B. Calder. Motivation for variable length intervals and hierarchical phase behavior. In *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-05)*, pages 135–146, Washington, DC, USA, 2005. IEEE Computer Society.
- [5] J. Lau, E. Perelman, and B. Calder. Selecting software phase markers with code structure analysis. In *CGO '06: Proceedings of the International Symposium on Code Generation and Optimization*, pages 135–146, Washington, DC, USA, 2006. IEEE Computer Society.
- [6] J. Lau, J. Sampson, E. Perelman, G. Hamerly, and B. Calder. The strong correlation between code signatures and performance. In *Proceedings of the 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-05)*, pages 57–67, Washington, DC, USA, 2005. IEEE Computer Society.
- [7] J. Lau, S. Schoenmackers, and B. Calder. Transition phase classification and prediction. In *HPCA*, pages 278–289. IEEE Computer Society, 2005.
- [8] H. Patil, R. S. Cohn, M. Charney, R. Kapoor, A. Sun, and A. Karunanidhi. Pinpointing representative portions of large intel®itanium®programs with dynamic instrumentation. In *The Proceedings of the 37th Annual International Symposium on Microarchitecture (MICRO-37 2004)*, 4-8 December 2004, Portland, OR, USA, pages 81–92. IEEE Computer Society, 2004.
- [9] C. Pereira, J. Lau, B. Calder, and R. Gupta. Dynamic phase analysis for cycle-close trace generation. In *CODES+ISSS '05:*

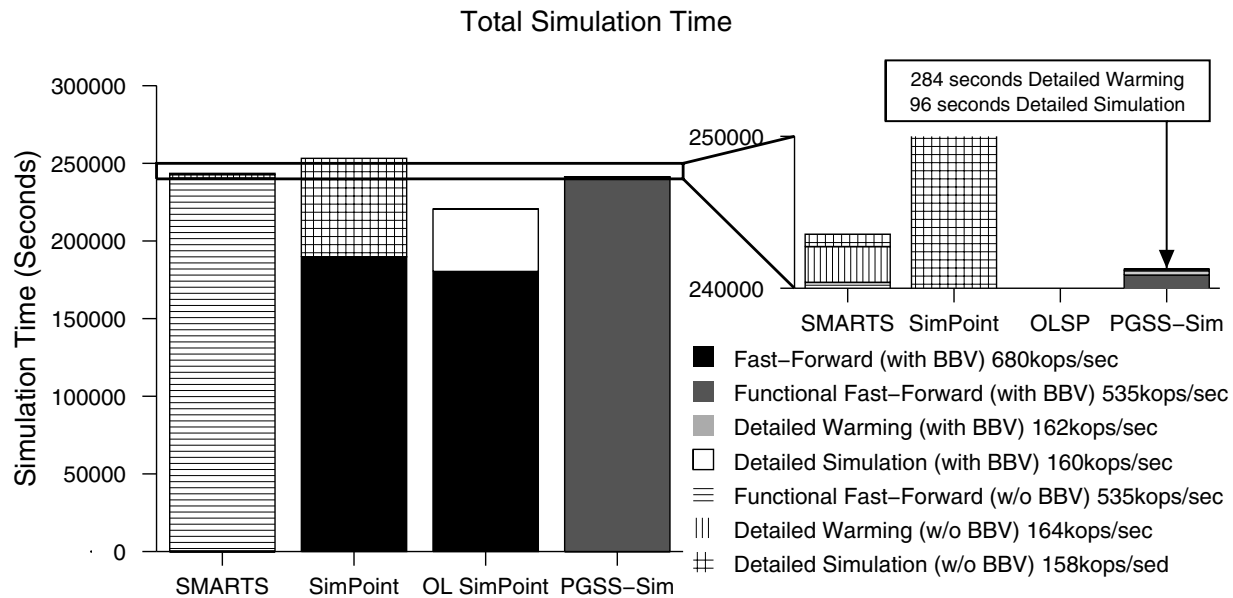


Figure 13. The total simulation times for the ten tested *Spec2000* benchmarks for SMARTS, SimPoint (10 clusters of 100 million operations), On-Line SimPoint (100 million-operation samples, $.1\pi$ threshold), and PGSS-Sim (1-million operation sample period, $.05\pi$ threshold). This data does not account for off-line analysis or the use of any checkpointing.

Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pages 321–326, 2005.

- [10] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *PACT '03: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, page 244, Washington, DC, USA, 2003. IEEE Computer Society.
- [11] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically characterizing large scale program behavior. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 45–57, 2002.
- [12] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 336–349, New York, NY, USA, 2003. ACM Press.
- [13] Standard Performance Evaluation Corporation. The SPEC CPU 2000 benchmark suite, 2000.
- [14] Standard Performance Evaluation Corporation. The SPEC CPU 2006 benchmark suite, 2006.
- [15] T. Wensisch, R. Wunderlich, B. Falsafi, and J. Hoe. Simulation sampling with live-points. In *Proceedings of the 2006 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-06)*, pages 2–12, Washington, DC, USA, 2006. IEEE Computer Society.
- [16] R. E. Wunderlich, T. F. Wensisch, B. Falsafi, and J. C. Hoe. Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling. In *The Proceedings of the 30th International Symposium on Computer Architecture (ISCA 2003)*, 9-11 June 2003, San Diego, California, USA, pages 84–95, 2003.
- [17] R. E. Wunderlich, T. F. Wensisch, B. Falsafi, and J. C. Hoe. An evaluation of stratified sampling of microarchitecture simulations. In *The Proceedings of the Third Annual Workshop on Duplicating, Deconstructing, and Debunking (held in conjunction with ISCA-31)*, 2004.
- [18] J. J. Yi, S. V. Kodakara, R. Sendag, D. J. Lilja, and D. M. Hawkins. Characterizing and comparing prevailing simulation techniques. In *The Proceedings of the 11th International Conference on High-Performance Computer Architecture (HPCA-11 2005)*, 12-16 February 2005, San Francisco, CA, USA, pages 266–277. IEEE Computer Society, 2005.
- [19] J. J. Yi, H. Vandierendonck, L. Eeckhout, and D. J. Lilja. The exigency of benchmark and compiler drift: Designing tomorrow's processors with yesterday's tools. In *Proceedings of the 2006 International Conference on Supercomputing (ICS)*, pages 75–86, Cairns, Australia, 6 2006. ACM.