Simulation Points for SPEC CPU 2006

Arun A. Nair, Lizy K. John
Dept. of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX 78712 USA
{*nair, ljohn*}*@ece.utexas.edu*

***Abstract*— Increasing sizes of benchmarks make detailed simulation an extremely time consuming process. Statistical techniques such as the SimPoint methodology have been proposed in order to address this problem during the initial design phase. The SimPoint methodology attempts to identify repetitive, long, large-grain phases in programs and predict the performance of the architecture based on its aggregate performance on the individual phases. This study attempts to compare accuracy of the SimPoint methodology for the SPEC CPU 2006 benchmark suite with that of SPEC CPU 2000 and to study the large-grain phases in the two benchmark suites using the SimPoint methodology. We find that there has not been a significant increase in the number of simulation points required to accurately predict the behavior of the programs in SPEC CPU 2006, despite its significantly larger data footprint and dynamic instruction count. We also find that the programs in both benchmark suites have similar characteristics in terms of the number of phases that contribute significantly towards overall behavior, further emphasizing the similarity between the two benchmark suites with respect to the number of simulation points required for similar accuracies.**

I. Introduction

The SPEC CPU 2006 suite, the latest SPEC CPU benchmark suite provided by the Standards Performance Evaluation Corportation [7], consists of 12 integer and 17 floating point programs. The programs have significantly larger dynamic instruction counts and data footprint than the earlier SPEC 2000 benchmark suite [2]. The increased runtime of the programs due to the combined effect of the increased data footprint and dynamic instruction count make detailed simulation on a software-based micro-architectural simulator impractical. The SimPoint technique was proposed by Sherwood et al. [3] as a means of reducing effective run time of programs without inducing significant errors. The technique takes advantage of the fact that programs in the SPEC suite exhibit long repetitive phases of execution. Therefore, simulation of phases that are representative of these repeating phases should suffice towards providing a reasonably accurate prediction of performance of the architecture. The methodology actually uses a proxy for the unique phases, called simulation points, the details of which shall be presented in Section II.

The accuracy of the SimPoint methodology depends on the number of unique phases that are specified to be uncovered. Too many simulation points could result in longer simulation time and/or resources, and too few would result in large errors. Since SPEC CPU 2006 has at least an order of magnitude larger number of dynamic instructions than SPEC CPU 2000, one might expect that it would have a larger number of unique phases and hence require larger number of simulation points. Unfortunately, it is hard to know how many simulation points would suffice for low error in prediction without doing a validation of the methodology. This validation process would involve running the program in its entirety and comparing the results thus obtained with those obtained through the use of the SimPoint methodology.

Since the process of identifying SimPoints is ISA independent, validation on an x86 machine would validate the process across any other ISA. We compare the number of simulation points required for producing accurate results with SPEC CPU 2006 with those obtained using SPEC CPU 2000 and compare the results. The major contributions of this work are:

1) Validation of SimPoint for SPEC CPU 2006. This will provide justification for the use of the technique to accelerate simulation using this benchmark suite.
2) Contrasting the number of unique phases required for accurate simulation of programs in SPEC CPU 2006 with that of SPEC CPU 2000. It is impossible to know how many simulation points are required for low error without such a study. Also, we study the similarity in the distribution of phases in terms of their contribution towards overall execution.

We expect that our work will provide a reference for architects to make an informed decision about the number of simulation points that would be needed to have low error while using the SimPoints methodology with SPEC CPU 2006.

II. Background

We use the PinPoints tool provided by Intel Corporation in order to identify the simulation points in the program. PinPoints uses the PIN tool to extract program characteristics from x86 binaries, which can then be used by the SimPoint tool. Thus, it eliminates the need for a functional simulator by allowing code to run directly on real hardware. The details of the SimPoint methodology and the PinPoints tool are presented below.

A. SimPoint

The SimPoint methodology provides a means of identifying and isolating unique phase behavior that exist in

many programs. A phase may be thought of as a region of execution when the program execution is stable - the program exhibits a relatively constant CPI, cache misses etc. The SimPoint methodology involves uncovering all phases in the dynamic execution stream, grouping similar phases together, and picking a representative phase from each group.

The first step in this process involves slicing the dynamic execution trace into chunks of a fixed size. Large slices may fail to capture certain short-lived phase behavior but can generate reasonably accurate results without the need to warm up the caches. Reduction in the size of slices makes each simulation point susceptible to errors due to cold cache misses, but will identify phases of shorter duration, and may therefore lead to better simulation accuracy.

The next step is to identify characteristics that can be used to measure similarity between slices. Towards this end, the SimPoints procedure creates Basic Block Vectors (BBV) for each slice. Each BBV contains the product of the number of times the basic block was executed and the number of instructions in that basic block. This is used to group similar slices together. A representative slice is then chosen from amongst them, which is called a simulation point or SimPoint. Simulation of these slices followed by a weighted addition of results obtained through this process may be used to predict the behavior of the entire program on the architecture. Clustering algorithms may be used to achieve this goal; however, highly dimensional data causes these algorithms to run very slowly. The basic block vectors are therefore normalized and random linear projection is used to reduce the dimensionality of the basic block vectors down to 15. K-means clustering is now used to form clusters of similar slices. K-means algorithm involves assigning a set of points randomly as cluster centers in the multi-dimensional space, assigning each slice to the closest center, and then iteratively assigning cluster centers to the centroid of each such cluster. The algorithm terminates once there is no change in cluster center position or a maximum count is reached. Since K-means algorithm requires the number of centers to be known a priori, the algorithm is run multiple times with multiple centers, and Bayesian Information Criterion (BIC) is used to pick the smallest value of K that produces the best fit for the clustering.

At the end of the process, a representative slice is picked from each cluster, which is called a simulation point. Each SimPoint is weighted based upon the size of the cluster it represents. Now detailed simulation needs to be done starting at the SimPoints, over the length of the slice. A weighted addition of the parameter of interest (such as CPI) is used to predict the parameter of a complete simulation. Since the simulation points are independent of one another, they can be run in parallel, resulting in a significant reduction of simulation time.

Sherwood et al. use slices of 100 million instructions, with up to 10 simulation points in order to validate the procedure using the SimpleScalar simulator running SPEC CPU 2000 Alpha binaries, which resulted in an average IPC error of 3% [3]. The process was also verified for L1 and L2 cache miss rate and branch prediction accuracy.

B. PinPoints

PIN [6], [9] is a dynamic instrumentation tool targeted at Intel®Xscale, x86 and IA64 platforms. It provides a rich set of APIs that can be used to study various characteristics of program behavior at the level of the ISA. Since the program is instrumented dynamically and runs natively on hardware, using PIN provides orders of magnitude of speedup over a functional simulator. PinPoints [8] is a tool built on top of PIN that applies the SimPoint technique to programs compiled for the aforementioned ISAs. Since the programs that constitute SPEC CPU 2006 has dynamic instruction counts of over one trillion, a detailed microarchitectural simulator would take several months, if not more than a year to simulate a complete run of each program. In order to check the applicability of SimPoints for SPEC CPU 2006, we would require a detailed run of each program in the benchmark suite, which is quite infeasible. We therefore use the PinPoint technique to validate SimPoints as a technique for evaluation of new architectural features through detailed simulation.

PinPoints provides tools to analyze the dynamic instruction trace of the program and produce a basic block vector, which can be used to identify simulation points using the SimPoint methodology. The identified simulation points can then be used to simulate various microarchitectural features such as cache hierarchies and branch predictors, using PIN. These simulation points can also be used in a detailed x86 simulator to predict parameters such as CPI, cache miss rates and branch predictor accuracies. Using a maximum of 10 simulation points with each slice of size 250 million instructions, they reported CPI errors of less than 10%, on multiple processor configurations. The procedure was also verified by comparing the results obtained by executing identified simulation points on a detailed microarchitectural simulator with a complete run on real hardware. In this case too, the procedure compared favorably, with low errors on most programs.

III. Evaluation Setup and Methodology

The programs in the benchmark suite are first profiled using the PinPoints tool in order to generate the BBVs, which are then processed using the SimPoint methodology to obtain the simulation points and their corresponding weights. We compare the errors obtained from predicting CPI, L1 and L2 cache misses using the SimPoints methodology for SPEC CPU 2006 and SPEC CPU 2000, for the same maximum number of simulation points. In order to do this, we would need data on CPI, L1 and L2 cache misses per kilo-instruction (MPKI) for the complete run of each program in the benchmark suites as compared to the weighted aggregate of these respective metrics measured at each simulation point, over the length of a SimPoint slice. We also need to select a maximum possible number of simulation points for the K-means clustering algorithm. The smaller the maximum, the greater is the potential error due to the shortage of

representative simulation points. The program could have had many repetitive phases, but by selecting a low maxK, we may force it to compromise in its selection of representative points. In order to minimize this effect, we fix the maximum number of simulation points (maxK value) to 30. As can be seen in Table I and Table II, most programs are well short of the maximum number of simulation points.

We use a SimPoint slice of 100 million instructions, which is the same size as that used by Sherwood et al.[3] while evaluating the SimPoints methodology. The rationale behind the use of large slices is that they minimize the effect of cold misses in the cache for each simulation point and therefore, warming up of the caches prior to execution of each simulation point is unnecessary [12]. We use pfmon [10] to measure both the overall metrics as well as the metrics at each simulation point. Pfmon reads the performance monitoring counters of the processor. We run each program one at a time on an Intel®Pentium®4 machine with hyper-threading, running at a clock frequency of 2.8 GHz. It is equipped with a 16kB, 8-way set associative L1-D cache and 1MB, 8-way set-associative L2 cache, both having 64 byte lines. We use Linux kernel 2.6.23.1, patched and recompiled to support pfmon. The SPEC benchmark binaries are compiled using gcc 4.2 in base configuration. Since simulation of all 55 program-input combination of SPEC CPU 2006 programs is impractical, we use the results of clustering techniques applied to SPEC CPU 2006 by Phansalkar et al. [14] to pick one representative input for the program. A complete run of each benchmark program is first performed, and the aggregate information of instructions retired and the number of clock cycles consumed in the process is directly reported using pfmon. Then a second run is performed in which the corresponding counter is sampled using pfmon is sampled every 100 million instructions. The difference between the value at the start of the simulation point and the subsequent sample is used to compute the number of clock cycles elapsed during that simulation slice, taking counter overflows into account. This process effectively provides a warmed-up cache for each simulation point. However, since a large simulation point is not significantly affected by the state of the cache, we do not expect our results to be far from what would have been obtained using a cold cache. A similar approach was used by Patil et al. [8] in their work to validate PinPoints. We run each of the above simulations three times and pick the lowest value obtained, in order to minimize errors due to interference from other processes that may have been running at the time. The process is repeated to obtain the L1 and L2 Misses Per Kilo Instructions (MPKI).

IV. Results

We begin by analyzing the differences between the two benchmark suites in terms of the number of simulation points that were produced. This data is useful in order to estimate the amount of additional simulation time, if any, required to run a program from benchmark suite as compared to the other. We then contrast the number of simulation points that have significant weights associated with them, and the number of such simulation points that contribute 90% of the total weights. This analysis will provide useful information on the nature of phase behavior in programs of both benchmark suites, and their differences, if any. We finally compare the error for the two benchmark suites in order to determine the need for additional simulation points for SPEC CPU 2006, considering its vastly increased size over SPEC CPU 2000.

A. Comparison of Simulation Points for SPEC CPU 2006 and SPEC CPU 2000

The number of simulation points and their weights, generated for each program in SPEC CPU 2006 and SPEC CPU 2000 can be used to compare their overall behavior. Table I lists the total number of simulation points and the total number of dynamic instructions for the SPEC CPU 2006. Most of the programs had less than 21 simulation points, with only four exceeding this value. The table also contains the dynamic instruction count as obtained using pfmon. The number of simulations points and number of instructions may vary depending on which input file is used for the programs.

We analyze the weights of the simulation points to evaluate the phase behavior of programs. A large weight on a simulation point would imply that the program spends a longer fraction of time in that phase of execution. Calculix has a dominant simulation point that accounts for 42% of the execution, and four simulation points together account for almost 80% of the execution. This indicates that there is very low diversity in the behavior of this program. CactusADM has an even more dominant simulation point, accounting for 73% of the total weight, and three simulation points accounting for over 90% of the execution. For such programs, a smaller maxK value may yield acceptable results. Programs such as soplex, namd, milc, zeusmp and leslie3d have a fairly regular distribution of weights and hence need more number of simulation points for better accuracy. Many programs, such as perlbmk, lbm and cactusADM have a substantial number of simulation points with very low weights. While the existence of these phases indicate diversity in program behavior, they do not contribute significantly to the overall CPI. Such programs can be simulated accurately using a fewer number of simulation points. For instance, cactusADM has 21 simulation points, but only 10 of them individually account for more than 1% of the execution, while the rest combined contribute approximately 0.5% to the total. This analysis indicates that we can reduce the number of simulation points used by the programs and in many cases, achieve high accuracies. Only programs that have a regular distribution of phases, so as to not have a dominant phase would need additional phases. This information can be used to adaptively fix the number of simulation points that would yield acceptable results during simulation, saving on resources and/or total simulation time.

While programs may have a large number of phases, many have only a few dominant phases. These phases contribute most significantly towards the overall behavior of the program and hence are assigned the highest weights in the

SimPoint methodology. It may be possible that most SPEC CPU 2000 programs have very few such dominant simulation points, and hence there is no significant loss in accuracy even if the more infrequent phases are ignored, whereas CPU 2006 may not. We would therefore need to address the question whether a reduction in the number of simulation points (maxK) would result in different behavior as far as errors are concerned. A significant discrepency in the distribution of weights in one benchmark suite over another would suggest that one suite requires more simulation points than the other.

We therefore analyze the weights of programs in CPU 2000 suite, much in the same way as we did earlier for SPEC CPU 2006. We find that the overall trends are very similar in both. There is a similar mix of some programs having few, dominant simulation points, and others having simulation points with similar weights. For instance, apsi has a very dominant simulation point which contributes 50% of the total weight. Six simulation points in sixtrack contribute over 98% of the execution. However, the other programs have weights that are close to one another. Programs such as bzip-graphic and equake have a number of simulation points with relatively similar weights. Hence these programs would have higher errors if the number of simulation points were reduced.

In order to gauge the number of phases that contribute significantly to the overall program behavior, we identify the 10 most dominant simulation points of the benchmarks in both suites, based on their weights. The higher the cumulative weight for these simulation points, the greater the influence of the 10 simulation points on program behavior, and potentially, the better the accuracy when maxK is reduced to 10. We find that 10 simulation points cover at least 70% of the total weight for 17 out of a total of 25 SPEC CPU 2000 programs evaluated. 8 of these programs have over 90% of their total weights covered by 10 simulation points. A similar analysis for CPU 2006 reveals that 22 out of a total of 28 programs cover at least 70% of the total weights, and that 10 of these have over 90% of their total weight represented by 10 simulation points. For both benchmark suites, majority of the programs ranged between 75% and 85% of the maximum, for 10 simulation points. This suggests that both programs have similar distribution of weights for simulation points. This result is important because it suggests that reducing the number of simulation points will not decrease accuracy on one benchmark suite versus the other. It also provides us with a first-order approximation of the effect of reducing the number of simulation points without actually having to simulate for multiple max-K values.

We also try to identify the number of simulation points that combined, constitute 90% of the overall program execution. The program spends a significant amount of time in these phases. A fewer number of simulation points required to achieve this number would indicate fewer significant phases of program execution. We find that the average number of simulation points required to achieve 90% of the total weight in CPU 2006 suite is 13.07, with a standard deviation of 4.3, whereas that for CPU 2000 is 13.4, with a standard deviation

TABLE I

NUMBER OF SIMULATION POINTS, NUMBER OF SIMULATION POINTS AMOUNTING TO 90% OF TOTAL EXECUTION AND INSTRUCTION COUNT FOR SPEC CPU 2006

Benchmark	Simulation Points	90 percentile Points	Instructions (billions)
400.perlbench-splitmail	21	12	756.9
401.bzip2-combined	17	13	371.92
403.gcc-scilab	17	9	68.57
429.mcf	14	9	464.98
445.gobmk-trevord.tst	18	13	359.52
456.hmmer-retro.hmm	17	15	2472.91
458.sjeng	16	12	2654.13
459.gemsFDTD	20	12	308.88
462.libquantum	22	15	4534.27
464.h264ref-sss_encoder_main	20	14	3289.98
471.omnetpp	9	6	787.08
473.astar-rivers.cfg	8	6	961.44
483.xalancbmk	18	13	1401.34
410.bwaves	22	10	2780.95
416.games-triazolium	15	11	3717.7
433.milc	23	18	1649.57
434.zeusmp	26	19	2273.56
435.gromacs	20	19	2267
436.cactusADM	21	3	3115.92
437.leslie3d	22	20	4745.74
444.namd	26	18	3293.89
447.dealII	21	14	2809.95
450.soplex-ref.mps	21	17	414.17
454.calculix	10	7	8499.78
453.povray	20	15	1287.36
465.tonto	20	15	3002.2
470.lbm	21	12	1567.55
482.sphinx	20	16	3135.75
Average	**18.75**	**13.07**	**2249.75**

TABLE II

NUMBER OF SIMULATION POINTS, NUMBER OF SIMULATION POINTS AMOUNTING TO 90% OF TOTAL EXECUTION AND INSTRUCTION COUNT FOR SPEC CPU 2000

Benchmark	Simulation Points	90 percentile Points	Instructions (billions)
176.gcc-scilab	18	11	38.51
176.gcc-166	23	14	21.29
164.gzip-graphic	27	21	71.47
164.gzip-source	14	10	54.17
175.vpr-place	15	11	111.86
175.vpr-route	23	15	85.63
300.twolf	20	14	290.93
186.crafty	16	13	216.96
181.mcf	12	8	48.80
253.perlbmk	16	10	94.87
256.bzip-source	20	15	87.08
256.bzip-graphic	23	20	117.28
197.parser	13	10	281.77
254.gap	18	12	54.17
179.art-1	15	12	113.55
179.art-2	12	10	117.29
173.applu	25	18	528.82
188.ammp	24	13	386.60
200.sixtrack	9	5	936.54
183.equake	30	23	149.67
301.apsi	20	11	602.69
171.swim	22	20	249.89
172.mgrid	22	17	523.77
168.wupwise	23	9	490.19
177.mesa	18	13	317.34
Average	**19.12**	**13.4**	**239.65**

of 4.4. The values for each benchmark can be found in Table I and II under the 90 percentile points column. Our analysis suggests that not only are the number of simulation points similar, but the weights of these phases are also similar in distribution. Hence, a reduced number of simulation points would result in inaccuracies not dissimilar to those observed for CPU 2000.

B. Comparison of Errors in CPI and Cache Misses

Figures 1(a) and 1(b) plot the CPI obtained from a complete run as compared to the results obtained from the weighted addition of the CPI of each simulation point. The highest error for SPEC CPU 2006 is obtained for GemsFDTD, at 10.24%, and the average error is 2.45%. The highest error for SPEC CPU 2000 is 10.7%, obtained for gcc-166. The average error obtained is 2.15%. This validates the efficacy of the SimPoints procedure for SPEC CPU 2006. We may conclude that the errors for both benchmark suites are very similar, for the same maximum number of simulation points. Despite the large increase in the dynamic instruction count of SPEC CPU 2006 programs, an increase in the number of simulation points over SPEC CPU 2000 is unnecessary.

Figure 1(c) plots the error in misses per kilo-instructions (MPKI) for L1 cache, using the SimPoints methodology, for CPU 2006. The average error obtained for L1 MPKI on SPEC CPU 2006 is 3.55%. This compares favorably with the average error of 6.51% for CPU 2000, as shown in Figure 1(d). The large average error is predominantly due to a relatively higher error of 31.57% on gzip-graphic. In absolute terms, this is a result of an error of 9 misses per kilo-instructions. The graphic input set is also responsible for the third-highest error value of 18% in bzip2. The error in MPKI here is 2.95. Barring these exceptions, the overall L1 MPKI results are very similar.

In the average error between the predicted and actual MPKI for SPEC CPU 2006 is 12.62%, as shown in Figure 1(e). The highest error in L2 cache MPKI is for gobmk (115%). When compared to SPEC CPU 2000 (Figure 1(f)), we obtain an average error of 23.77% and a maximum error of 271% for wupwise. The percentage values appear to be high, but this is largely due to the fact that L2 MPKI for all programs tends to be a very small value, and in many cases, zero. Hence, depending on the actual MPKI, a small deviation may show up as a large percentage, or vice versa. This is indeed true for both gobmk and wupwise. In absolute terms, the highest error is for xalancbmk (0.41) and art-2 (2.9) for SPEC CPU 2006 and SPEC CPU 2000 respectively. Since misses do not necessarily cause stalls in superscalar processors, simulation points with higher inaccuracy in predicting L2 misses may not necessarily also have higher error in CPI prediction . Clearly, the reverse of this is also true – not all programs with high cache accuracy will provide high accuracy with CPI. Since the simulation point cannot possibly be identical to all other phases it is designated to represent, differences in instruction mix and data access contribute to errors.

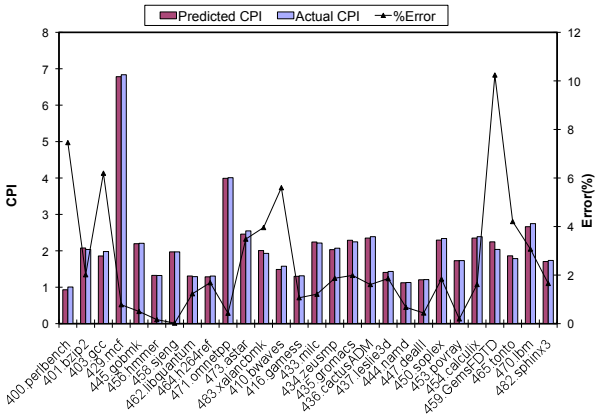
These results are interesting, as they indicate that even though there has been orders of magnitude increase in the dynamic instruction counts and data footprints of programs in SPEC CPU 2006 over CPU 2000, for the same number of maximum simulation points, the error rates are very similar. It suggests that the number of phases in the programs that constitute the benchmark suite have not changed, in spite of an increase in the data footprint and dynamic instruction count. Even for programs with dynamic count exceeding 3000 billion instructions, the total number of simulation points required for generating accurate results remains small. Calculix represents a good example for the repetitive phase behavior for which the SimPoints technique was devised: it has the highest dynamic instruction count of approximately 8500 billion instructions, and yet can be represented with just 10 simulation points with an error of only 3% for CPI. The fact that the number of simulation points required for accurate simulation remains practically unchanged is important for performance evaluation studies using the SimPoint methodology. Researchers may use these results to make an educated choice on the number of simulation points required for the SPEC CPU 2006 program that is of interest.

It is also interesting to note that overall, the CPI in both benchmark suites is similar. This is, in part, due to the fact that the L2 miss rates in both benchmark suites are very similar as well. A detailed analysis of this is beyond the scope of this paper. This observation seems to agree with the findings of Gove [1], who performed a comparative study of the working set sizes of SPEC CPU 2006 and CPU 2000. Gove notes that the increase in memory footprint for integer programs had not resulted in a proportionate increase in the working set size (WSS) of all programs in the SPEC CPU 2006.

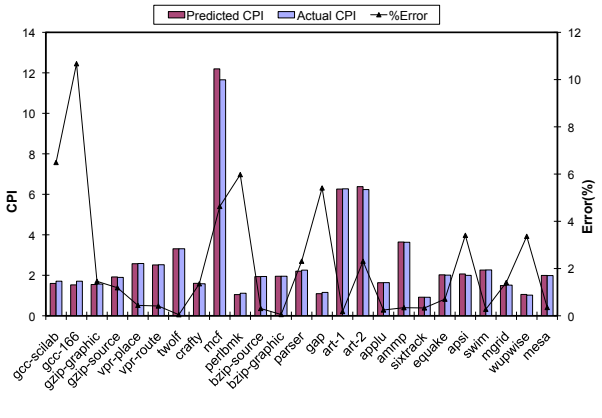
We may therefore conclude that SPEC CPU 2006 produces similar accuracies with similar number of simulation points, as compared to SPEC CPU 2000. Despite the significantly larger run-time for programs in SPEC CPU 2006 suite, our study indicates that there is no significant increase in simulation time, if the SimPoint methodology is adopted.

V. Related Work

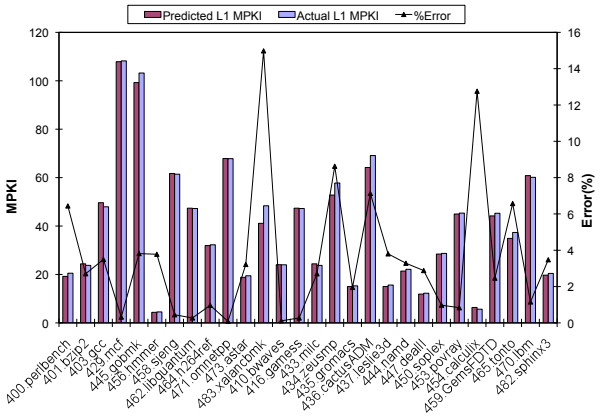
Sherwood et al [3], [11] presented the SimPoints technique and validated it for SimpleScalar using Alpha binaries. Patil et al [8] use Pin on IA64 to generate BBVs that may be used to identify simulation points using the SimPoint methodology. Both these have been covered in detail in Section II. Therefore, for the sake of brevity, we shall not repeat it here. Some comparative studies between the two benchmark suites have been done. Henning presented studies on the memory footprint of SPEC CPU 2000 [13] and SPEC CPU 2006 [2], in which it was noted that SPEC CPU 2000 was designed to target systems with 256MB of memory, whereas SPEC CPU 2006 targets a memory footprint of 900MB. Gove [1] analyzes the working set size (WSS) of SPEC 2006 programs and concludes that overall, the WSS of SPEC CPU 2000 benchmarks is not significantly different from that of SPEC CPU 2006. He notes that with



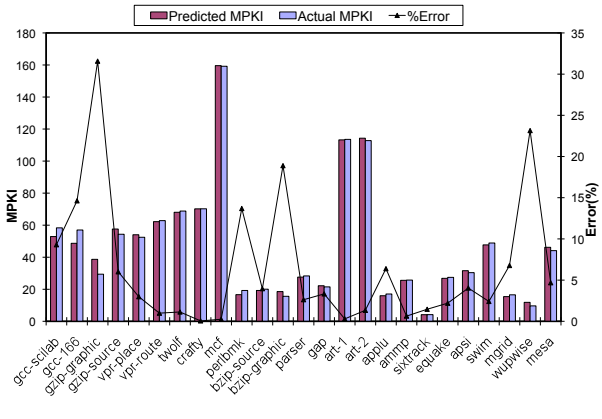
(a) CPI measurements for SPEC CPU 2006.



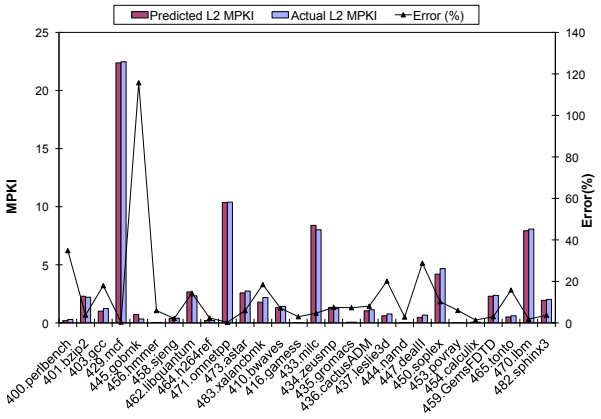
(b) CPI measurements for SPEC CPU 2000.



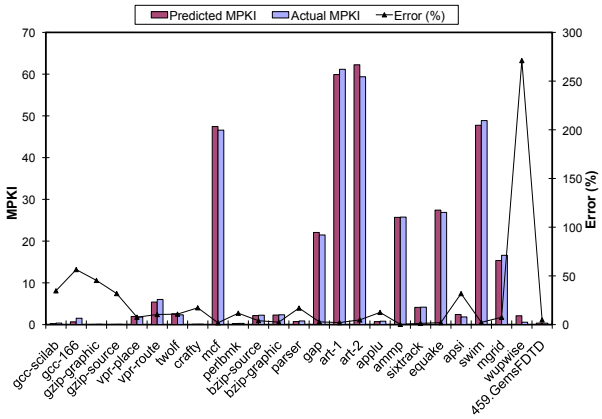
(c) L1 MPKI measurements for SPEC CPU 2006



(d) L1 MPKI measurements for SPEC CPU 2000



(e) L2 MPKI measurements for SPEC CPU 2006



(f) L2 MPKI measurements for SPEC CPU 2000

Fig. 1. **Comparison of CPI, L1 MPKI, L2 MPKI and their measurement error, obtained from the weighted aggregate of simulation point measurements (Predicted) and the measurements of a complete run (Actual).**

the exception of mcf in CPU 2006, all integer programs in both suites have a WSS of less than 256MB. However, for floating point programs, there is significant increase in the WSS, with around 25% exceeding 256MB. He also notes that some floating point programs have very low working set sizes. We are unaware of any other comparative studies involving SPEC CPU 2000 and 2006.

VI. Summary

In this work, we apply the SimPoints procedure to SPEC CPU 2006, using PinPoints. Our results indicate that, for a maxK of 30, with slice size of 100 million instructions, we observe a low average CPI error of 2.45% and a maximum error of 10.24%. For most programs, the CPI error is well below 3%, indicating that SimPoints can be used to accelerate simulation of single threaded benchmarks on detailed simulators. For a maxK of 30, no program produced more than 26 simulation points, and all but one produced less than 23 simulation points. This fact, along with the the low CPI error indicates that CPU 2006 exhibits long repetitive phases, much like its predecessor, the SPEC CPU 2000 benchmark suite. We compare the results of the SimPoints methodology applied to SPEC CPU 2000 with the results obtained from SPEC CPU 2006, and find that the CPI error percentage is similar, for the same maxK and slice size. Furthermore, we note that the number of simulation points produced are also similar. This indicates that there is no significant increase in the number of unique phases in the new benchmark suite and hence additional simulation points are not required to adequately capture phase program behavior. From a simulation standpoint, this means that the overall time taken to predict performance using either benchmark remains the same, since we have to execute a similar number of fixed-size slices.

We may therefore conlude that while there are some differences in the working set size and memory footprint for many programs in the two benchmark suites, the fundamental behavior of the programs insofar as repetitive phase behavior is concerned, remains largely unchanged. This work provides the scientific community with sufficient information to decide on the number of simulation points that would be necessary for accurate simulation of a program in the SPEC CPU 2006 suite.

VII. Acknowledgements

We would like to thank the anonymous reviewers for their comments and feedback. We would also like to thank the members of the Laboratory of Computer Architecture for their feedback for improving this manuscript. This work was supported in part through the NSF award number 0702694.

References

[1] Gove, D. 2007. CPU2006 working set size. *SIGARCH Comput. Archit. News 35, 1 (Mar. 2007)*, 90-96.

[2] Henning, J. L. 2007. SPEC CPU2006 memory footprint. *SIGARCH Comput. Archit. News 35*, 1 (Mar. 2007), 84-89

[3] Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the 10th international Conference on Architectural Support For Programming Languages and Operating Systems* (San Jose, California, October 05 - 09, 2002). ASPLOS-X. ACM, New York, NY, 45-57.

[4] Burger, D.C. and Austin, T.M.. The SimpleScalar tool set, version 2.0. Technical Report CS-TR-97-1342, University of Wisconsin, Madison, June 1997.

[5] Open Source Development Labs, *Database Test 2*, `http://www.osdl.org/lab_activities/kernel_testing/osdl_database_test_suite/osdl_dbt-2`, 2007.

[6] PIN home page: `http://rogue.colorado.edu/Pin/`

[7] Standard Performance Evaluation Corporation (SPEC) website, `http://www.spec.org/`

[8] Patil, H., Cohn, R., Charney, M., Kapoor, R., Sun, A., and Karunanidhi, A. 2004. Pinpointing Representative Portions of Large Intel®Itanium®Programs with Dynamic Instrumentation. In Proceedings of the 37th Annual IEEE/ACM international Symposium on Microarchitecture (Portland, Oregon, December 04 - 08, 2004). *International Symposium on Microarchitecture.* IEEE Computer Society, Washington, DC, 81-92.

[9] Luk, C., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. 2005. Pin: building customized program analysis tools with dynamic instrumentation. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation* (Chicago, IL, USA, June 12 - 15, 2005). PLDI '05. ACM, New York, NY, 190-200.

[10] Perfmon2 home page `http://perfmon2.sourceforge.net/`

[11] Sherwood, T., Perelman, E., and Calder, B. 2001. Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications. In *Proceedings of the 2001 international Conference on Parallel Architectures and Compilation Techniques* (September 08 - 12, 2001). PACT. IEEE Computer Society, Washington, DC, 3-14.

[12] Conte, T. M., Hirsch, M. A., and Menezes, K. N. 1996. Reducing State Loss For Effective Trace Sampling of Superscalar Processors. In *Proceedings of the 1996 international Conference on Computer Design, VLSI in Computers and Processors* (October 07 - 09, 1996). ICCD. IEEE Computer Society, Washington, DC, 468-477.

[13] Henning, J.L., SPEC CPU2000 Memory Footprint, `http://www.spec.org/cpu2000/analysis/memory`

[14] Phansalkar, A., Joshi, A., and John, L. K. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In Proceedings of the *34th Annual international Symposium on Computer Architecture* (San Diego, California, USA, June 09 - 13, 2007). ISCA '07