Multi-level Phase Analysis for Sampling Simulation

Jiaxin Li*†, Weihua Zhang*†, Haibo Chen‡, Binyu Zang*
*Parallel Process Institute, Fudan University
Email: {lijiaxin, zhangweihua, byzang}@fudan.edu.cn
†The State Key Lab of ASIC & System, Fudan University
‡Institute of Parallel and Distributed Systems, Shanghai Jiaotong University
Email: haibochen@sjtu.edu.cn

***Abstract*—Extremely long simulation time of architectural simulators has been a major impediment to their wide applicability. To accelerate architectural simulation, prior researchers have proposed representative sampling simulation to trade small loss of accuracy for notable speed improvement. Generally, they use fine-grained phase analysis to select only a small representative portion of program execution intervals for detailed cycle-accurate simulation, while functionally simulating the remaining portion. However, though phase granularity is one of the most important factors to simulation speed, it has not been well investigated and most prior researches explore a fine-grained scheme. This limits their effectiveness in further improving simulation speed with the requirement of increasingly complex architectural designs and new lengthy benchmarks.**

In this paper, by analyzing the impact of phase granularity on simulation speed, we observe that coarse-grained phases can better capture the overall program characteristics with a less number of phases and the last representative phase could be classified in a very early program position, leading to fewer execution internals being functionally simulated. By contrast, fine-grained phases usually have much shorter execution intervals and thus the overall detailed simulation time could be reduced. Based on the above observation, we design a multi-level sampling simulation technique that combines both fine-grained and coarse-grained phase analysis for sampling simulation. Such a scheme uses fine-grained simulation points to represent only the selected coarse-grained simulation points instead of the entire program execution, thus it could further reduce both the functional and detailed simulation time. Experimental results using SPEC2000 show such a framework is effective: using the SimPoint method as baseline, it can reduce about 90% functional simulation time and about 50% detailed simulation time. It finally achieves a geometric average speedup of 14.04X over SimPoint with comparable accuracy.

I. Introduction

Architecture simulators are indispensable in evaluating new architectural designs. Despite their advantages of low cost and high flexibility, extremely long simulation time limits their wide applicability. Therefore, reducing simulation time has become a crucial issue in designing such simulators [1]. Among prior solutions, representative sampling simulation techniques, which exploit cyclic (phase) behavior patterns in programs, have been shown to be extremely effective to reduce simulation time [1].

In a representative sampling simulation technique, a program's execution is first divided into non-overlapping intervals, each of which is a contiguous portion of program execution. Then, such intervals with similar behavior (e.g., similar IPC, and/or cache miss rates) are classified into the same phase. After the phase classification, a small representative portion of intervals (often referred to as *simulation points*) is selected based on their phase (i.e., cyclic) behavior. These selected simulation points are simulated in cycle-accurate detail, and the remaining portion is only functionally simulated, which is known as *fast-forward simulation*. The granularity (or the size) of the intervals is referred to as the *phase granularity* in this study.

978-3-9815370-0-0/DATE13/©2013 EDAA

Although sampling techniques have been widely used, they still need to be further improved to satisfy the increasing demand of emerging architectural designs. Nowadays, architectural designs have become more complex with more system configurations and design tradeoffs to be evaluated. This situation has been further exacerbated as new benchmarks have become increasingly more complex with longer execution time. For example, while the simulation time of SPEC2000 could be reduced by about 15X using the SimPoint method, the simulation time of SPEC2006 becomes about 10X longer than that of SPEC2000. *All these point to a dire need for further improvement in simulation speed while holding the simulation errors in check.*

For a representative sampling simulation method, the speed and accuracy are mainly affected by two key factors: phase granularity and the metrics used to classify phase behavior [2]. Phase granularity can be either fine-grained or coarse-grained. Since loops are the most time-consuming part of a program execution, and an iteration of a loop is a repetitive execution of the loop body with a similar behavior, a loop iteration corresponds roughly to an interval mentioned above. The average size of iteration in outmost loops of all SPEC2000 benchmarks is about 2,000M (million) instructions. Hence, in this study, we consider an interval size in the range of 1M to 1,000M instructions as fine-grained, whereas, an interval size in the range of 1,000M instructions and above as coarse-grained. The metrics are critical program characteristics that identify the repetitive behavior of a program execution. They could be working set oriented, e.g. basic-block vectors (BBVs) [3][4]), or data access oriented, e.g. memory reuse distance [5].

Although the phase granularity has been one of the most important factors in a representative sampling simulation method [2], it has not been well investigated. Until now, it is still unclear which granularity is a better choice and

how to exploit the advantages of different granularities. Most researchers simply applied a fine-grained strategy because each of the selected intervals requiring time-consuming detailed simulation could be much smaller compared to a coarse-grained scheme.

In this paper, we present a comprehensive study on the granularity of representative sampling techniques, and provide some new insights into the choice of a phase granularity: 1) Coarse-grained phases can better capture the overall program characteristics with a less number of phases and the last representative phase could be classified in a very early program position, leading to fewer execution internals being functionally simulated. 2) Fine-grained phases usually have much shorter execution intervals and thus the overall detailed simulation time could be reduced.

Based on the above observation, we design a multi-level sampling simulation technique that combines both coarse-grained and fine-grained phase analysis for representative sampling simulation. We have designed and implemented a multi-level phase analysis framework. In this framework, we first design a coarse-grained approach called COASTS (COarse-grained Accurately Sampling Technique for Simulators) that expands the size of each simulation point, i.e., making it more coarse-grained. After selecting the simulation points through COASTS, we apply a fine-grained sampling method to further re-sample the selected coarse-grained simulation points. Such a multi-level scheme has the following advantages:

- Compared to fine-grained phases, the coarse-grained ones can better catch the overall characteristics and hide instant fine-grained changes, which could lead to fewer coarse-grained phases and the last representative phase could be classified in a very early program position. As a result, the coarse-grained approach could be used to reduce the functional simulation time, which has become one of time-dominant portions in a fine-grained representative sampling simulation method.
- After selecting the coarse-grained simulation points, we could further re-sample those coarse-grained ones through a fine-grained scheme. Since those fine-grained points are only used to represent the selected coarse-grained simulation points instead of the entire program execution, combining the coarse-grained approach can lead to less detailed simulation time compared to purely fine-grained sampling simulation methods.

Experimental results using SPEC2000 show that such a framework is effective. When using SimPoint, one of the most well-known representative sampling techniques, as the baseline, our approach can reduce about 90% functional simulation time and about 50% detailed simulation time. It achieves a geometric average speedup of 14.04X over the SimPoint method while still maintaining comparable accuracy. Based on our framework, the simulation of SPEC2000 can be finished in about half a day (about 15 hours).

II. Related Work

Since most programs exhibit repetitive behavior over many different metrics, phase analysis has been widely used to identify repetitive program behavior patterns for reconfigurable architectural design, dynamic optimization and simulation acceleration. In a phase analysis, *phase granularity* and the *metrics* used for phase selection are two key considerations [2]. Here, we give a brief overview on representative sampling simulation from these two perspectives.

As one of the most representative sampling techniques, SimPoint [3][4] divides program execution into non-overlapping fixed-length intervals. It uses BBVs as the metric for identifying phases. The method in [4] selected early simulation points through constraining the position of the last cluster (EarlySP metric). However, this method can only reduce some functional simulation time. Moreover, its setting configurations are sensitive to both benchmarks and interval length. In [6], Dhodapkar et al. showed that BBV performs better than other instruction-execution related metrics, such as the working set. Lau et al. [7] showed that, using loop frequency vectors as a metric performed almost as well as BBV in accuracy and could also yield fewer distinct phases that could cut down the number of simulation points. Although there are some discussions on granularity in [8], they do not exploit multi-level phase analysis for sampling simulation. Therefore, in their later work [9], they still exploited fine-grained phase behavior for sampling simulation. The software phase marker (SPM) [9] method is a BBV-based technique and selects fine-grained phases according to loop or procedure boundaries as variable-length intervals. It has about the same simulation time as 10M SimPoint with a comparable error rate.

III. Motivation

Although the phase granularity has been one of the most important factors in a representative sampling simulation method [2], until now, it is still unclear which granularity is a better choice for representative sampling simulation methods. In this section, we will discuss the limitation of only considering fine-grained phase and the advantages of applying multi-level phase analysis for sampling simulations. Then, we will give out our multi-level sampling simulation approach.

A. Limitation of Fine-grained Sampling

To reduce *detailed* simulation time, prior sampling techniques, such as SimPoint [3] and the SPM method [9], tend to choose fine-grained phases and to constrain the maximum interval size. However, when a program is spilt into finer-grained intervals, more sensitive changes in program behavior will be exposed, and more phases would be identified even its overall proportion in the entire program is very low (i.e., not so important). Consequently, more simulation points will be selected. Some of them will usually locate close to the end of the program execution. Hence, the simulation of program execution will need to be extended to a substantially larger portion of the program execution to cover the ending phases.

In such cases, even though using finer-grained phases could reduce the detailed simulation time of each simulation point, the total amount of simulated instructions could be substantially increased, which consequently increases the total simulation time (both functionally and detailed).

B. Observation and Motivation

To gain insight into possible solutions to further improve the efficiency of sampling simulation method, we studied the behavior of coarse-grained phases in SPEC2000 and found some interested characteristics, which could be exploited to further improve the efficiency of sampling simulation:

Reducing Functional Simulation Time: Based on the characteristics of coarse-grained phase distribution in SPEC2000, we found that the number of coarse-grained phases is very small. For SPEC2000, the average coarse-grained phase number is three and only four benchmarks are larger than three (4 for gzip, 6 for equake and 5 for fma3d). Furthermore, the position [1] of last coarse-grained simulation point is very small (i.e., early). For example, the average position for SPEC2000 is about 17% and only three benchmarks are larger than 30% (86% for gcc, 47% for art and 36%for bzip2). Based on this observation, we can select coarse-grained simulation points to optimize the functional simulation time in a sampling simulation approach.

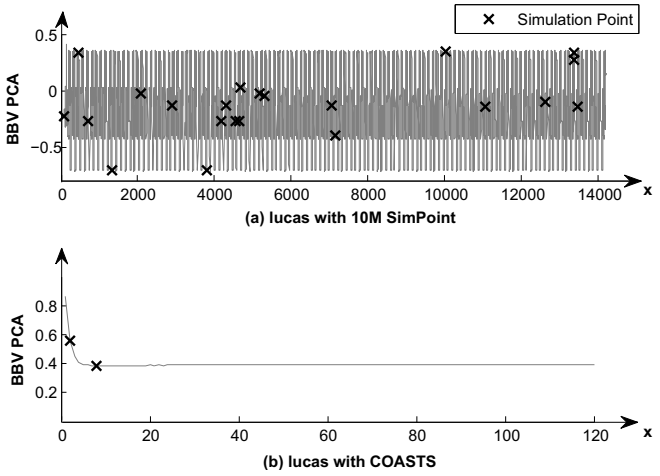

Fig. 1. This example illustrates how different granularities influence the selection of simulation points. The X-axis is the interval number (numbered according to its execution order), and the Y-axis is the value of first principal component of BBV in each interval. The check marks are the positions of the selected simulation points.

Here, we use a benchmark in SPEC2000, *lucas*, as an illustrating example. We collect the BBVs of each fixed-length interval of 10M instructions and that of our coarse-grained approach, respectively. Since BBVs are a kind of multi-dimensional data, we use Principal Components Analysis (PCA) [10] to extract their first principal component. Those PCA values are shown on the y-axis in Figure 1. The interval numbers are numbered according to their execution order, which are shown in the x-axis. As shown in the figure, the curves of the fine-grained method are very chaotic with violent changes. These thus lead to more phases being identified and more simulation points being selected close to the end of program execution (shown as the check marks in Figure 1(a)). In contrast, the curves of the coarse-grained approach are very smooth. Thus, much fewer simulation points at very early stages of the program execution are selected (shown in Figure 1(b)). As a result, a very large portion of program execution need not to be functionally simulated and the overall simulation time could be reduced.

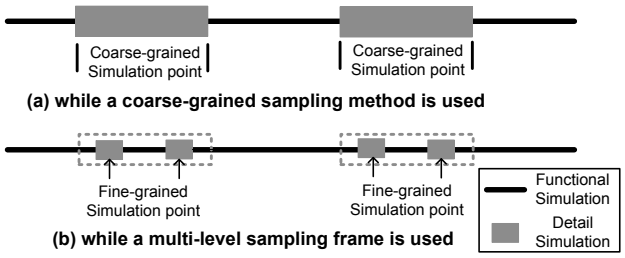

Fig. 2. This example illustrates how multi-level phase analysis framework works.

Reducing Detailed Simulation Time: Furthermore, as shown in Figure 2, after selecting the coarse-grained simulation points, we could apply a fine-grained sampling method again to those coarse-grained simulation points. Since those fine-grained simulation points are only used to represent the selected coarse-grained simulation points instead of the entire program execution, less fine-grained simulation points will be required, which can further optimize the detailed simulation time compared to pure fine-grained methods. Therefore, in a representative sampling simulation method, such a multi-level method could have the advantages of both coarse-grained and fine-grained approaches.

Since sampling methods tend to select the most representative parts to represent the entire execution of a program, it can guarantee the simulation accuracy to a certain level. Therefore, sampling twice in our multi-level sampling approach won't lead to more deviation. Our evaluation results confirm that the accuracy of our multi-level scheme is comparable to a fine-grained scheme.

IV. Design of Multi-level Sampling Simulation

In this section we will describe the design of our multi-level sampling framework. In this framework, we first select coarse-grained simulation points based on our coarse-grained sampling algorithm, which is referred as to COASTS. Then, we re-sample those coarse-grained simulation points to perform multi-level sampling via a fine-grained sampling algorithm.

A. Coarse-grained Sampling (COASTS)

For presentation clarity, we will refer to an interval instance as an *iteration instance*. Moreover, *a phase instance* is referred

[1] We define the position of an interval in a program to be the instruction number before its last instruction dividing the total instruction number in the program.

to as one occurrence of the phase. It could consist of multiple interval instances. As it could lead to a periodic behavior when the program is in a loop or in a recursive procedure, we refer them to as *cyclic program structures* in the rest of the paper.

In the first-level sampling stage, we try to obtain larger iteration sizes (or interval sizes). Hence, we tend to choose outer loops or shallow recursive calls to form coarse-grained intervals, instead of using inner ones with constrained interval sizes as in the SPM method [9]. Although a cyclic program structure exhibits some repetitive behavior, its iterations cannot be simply classified into a single phase because branches and memory accesses could lead to different dynamic behavior. Therefore, after choosing the appropriate level of a cyclic program structure, we classify the different iterations in the cyclic program structure into different phases based on the metrics. In this paper, we still choose BBVs [3] as the metrics for phase identification since BBV performs better than other instruction-execution related metrics, such as the working set as analyzed in [6]. Briefly, our approach requires the following three steps:

- Collection of Boundary Information: Currently, we use the similar method in [9][11] to collect the boundary information from dynamic profiling. Based on the profiling information, we first discard cyclic program structures with coverage less than 1%, since these contributed little to the final simulation results.
- Collection of Metrics Information: Metrics information, i.e., BBVs, is collected for each *iteration instance* of the selected cyclic program structures through a profiling stage. After the original information is collected, BBVs are randomly projected into their respective 15-dimension vectors. Such projections reduce computation complexity and storage requirement for the trace file. They also preserve behavior information for phase selection. Such a projection is widely used in prior techniques, such as SimPoint [3]. Then, BBV of each iteration instance are concatenated to form a signature vector. Such signature vectors are then normalized by having each element divided by the sum of all elements in the vector.
- Coarse-grained Sampling: After the metrics information is collected, we apply the *k-means clustering method* [3] for coarse-grained phase classification. Since the number of coarse-grained phase is small, the default K_{max} parameter for our coarse-grained phase clustering is three. Once the coarse-grained phases are classified, we choose the earliest instance of each coarse-grained phase as its representative (i.e., coarse-grained simulation point).

B. Fine-grained Sampling

Although our coarse-grained sampling method used in the first level sampling can effectively reduce the functional simulation time by decreasing the number of simulation points, the number of instructions in each simulation point can be increased. To further optimize the detailed simulation time in chosen coarse-grained simulation points, our multi-level sampling framework further re-sampling those coarse-grained simulation points in the second-level sampling via a fine-grained sampling method. If the size of a coarse-grained simulation point is larger than a threshold, we apply a fine-grained sampling method to re-sample it. Through selecting finer-grained simulation points within the coarser-grained simulation point, it can gain the advantages of both coarse-grained and fine-grained approaches.

V. Experimental Results

We evaluate our approach based on SimpleScalar tool set 3.0 [12] and SPEC2000 benchmarks with reference inputs for evaluation. The reasons we still choose SPEC2000 instead of SPEC2006 are that: First, The default parameters of the SimPoint method [3] are set based on the analysis on SPEC2000, so it is more reasonable to compare our approach with the SimPoint method using SPEC2000 benchmarks as inputs. Moreover, as analyzed in [13], the programs in SPEC2006 have similar phase behavior with those in SPEC2000, even the distribution of weights for simulation points.

TABLE I
CONFIGURATIONS

Part A: Base Configuration	
Parameter	**Value**
Out-of-Order Issue	8-way decode, issue, commit width
ROB/LSQ Entries	128/64
Registers	32 integer, 32 floating point
Functional Units	8-integer ALU, 4-load/store units, 2-FP adders, 2-integer MULT/DIV, 2-FP MULT/DIV
Instruction Cache	8k 2-way associative, 32 byte blocks, 1 cycle latency
Data Cache	16k 4-way associative, 32 byte blocks, 2 cycle latency
Unified L2 Cache	1Meg 4-way associative, 32 byte blocks, 20 cycle latency
Branch Predictor	Combined, 8K BHT Entries
Memory Latency	150, 10 cycle access (first, following)
Part B: Sensitivity Analysis Configuration	
Parameter	**Value**
Out-of-Order Issue	8-way decode, issue, commit width
ROB/LSQ Entries	128/64
Registers	32 integer, 32 floating point
Functional Units	6-integer ALU, 2-load/store units, 6-FP adders, 4-integer MULT/DIV, 4-FP MULT/DIV
Instruction Cache	32k direct mapping, 32 byte blocks, 1 cycle latency
Data Cache	128k 2-way associative, 32 byte blocks, 1 cycle latency
Unified L2 Cache	1Meg 4-way associative, 64 byte blocks, 23 cycle latency
Branch Predictor	Combined, 16K BHT Entries
Memory Latency	330, 20 cycle access (first, following)

To compare our results with those from SimPoint [3], the base machine configuration of the simulation is the same as that in [4] and [9], which is shown in Table I (Part A). To test the architecture sensitivity of our approach, we employ another architecture configuration shown in Table I (Part B). This configuration includes a larger cache size and a longer memory latency. CPI, L1 Cache hit rate and L2 Cache hit rate are used to measure the accuracy. The baseline data are collected from complete execution of each benchmark through the original version of sim-outorder. We then collect simulation results by simulating the simulation points selected by our approach and those by the current SimPoint release version (10M fixed length, $K_{max} = 30$). The reasons why we select 10M SimPoint are as follows: First, 10M SimPoint is the recommended interval length and is used in most of the SimPoint related papers; Second, although the SPM method

applied VLIs in it, we cannot compare our results with it directly because it is not implemented in current release of SimPoint. However, comparing with 10M SimPoint will not influence the final conclusions as the SPM method has about the same simulation time as 10M SimPoint with a comparable error rate [9]. Since the default K_{max} of 10M SimPoint is 30, we use 300M as the re-sampling threshold as described in Section IV-B(which is calculated as 10M*30=300M.)

In the following parts of this section, we will first present the experimental results of first-level sampling using COASTS alone and then illustrate the effect of our multi-level sampling framework via including second-level sampling.

A. Evaluation for COASTS

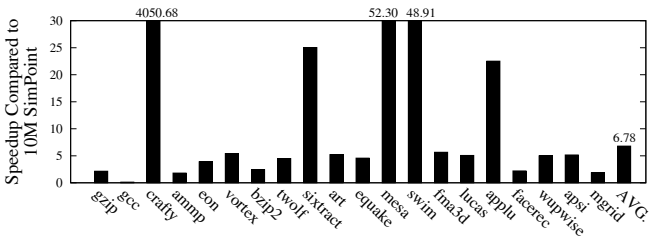

Fig. 3. Speedup of COASTS over SimPoint

TABLE II
DEVIATION COMPARISON

	Config A		Config B	
CPI	**AVG**	**Worst**	**AVG**	**Worst**
COASTS	0.93%	6.42%	0.66%	4.79%
SimPoint	1.43%	10.03%	2.35%	17.86%
Multi-level Sampling	1.88%	13.95%	1.53%	13.60%
L1 Cache Hit	**AVG**	**Worst**	**AVG**	**Worst**
COASTS	0.03%	0.75%	0.02%	0.89%
SimPoint	0.03%	0.25%	0.05%	0.26%
Multi-level Sampling	0.10%	1.89%	0.11%	1.90%
L2 Cache Hit	**AVG**	**Worst**	**AVG**	**Worst**
COASTS	0.13%	2.51%	0.18%	2.62%
SimPoint	1.32%	23.32%	2.83%	16.22%
Multi-level Sampling	1.35%	6.62%	1.92%	14.12%

Figure 3 and Table II show the speedup and the deviation, respectively, using COASTS approach and 10M SimPoint. Average results (**AVG**) use the geometric mean. Due to the space constraint, we only give the average deviation results and the deviation results in the worst cases. In Table II, the *AVG* is the average deviation results and the *Worst* is the worst deviation results. From the average results, our COASTS approach (first-level sampling) showed a speedup of $6.78X$ while maintaining comparable accuracy.

TABLE III
SIMULATION POINTS STATISTICS.

Algorithm	Mean Interval Size (inst.)	Mean Sample Number	Mean Detail (inst.%)	Mean Functional (inst.%)
COASTS	444M	1.6	0.37%	2.21%
10M SimPoint	10M	20.1	0.09%	93.76%
Multi-level Sampling	16M	7.3	0.05%	5.06%

To identify the root cause of the faster simulation, we compare a couple of metrics in Table III. In Table III, *Mean Interval size* is the geometric mean of interval lengths in different benchmarks, *Mean Sample number* is the geometric mean of the number of selected simulation points, *Mean Detail* is the total instructions simulated in detail divided by the total instructions and *Mean Functional* is the total instructions functionally simulated divided by the total instructions. Based on the above data, we can conclude that:

- Since the coarse-grained ones can better catch the overall characteristics and hide instant fine-grained changes, which leads to fewer coarse-grained phases and the last representative phase could be classified in a very early program position. In our study, even though benchmarks in SPECINT2000 are quite complex, fewer than three simulation points on average are needed. Coarse-grained samples lead to less total simulation time. As illustrate in Table III, our average sample size is larger (444M instructions, as opposed to 10M for the largest interval size used in SimPoint) and it leads to longer detailed simulation time (0.37% of total instructions simulated vs. 0.09% in 10M SimPoint). However, the functional simulation time is significantly reduced because fewer simulation points near the end of program execution are selected (2.21% of total instructions simulated vs. 93.76% in 10M SimPoint). A larger reduction of the proportion of the simulated instructions leads to a higher performance improvement.
- Regardless of a coarse-grained or a fine-grained sampling method, the variable length interval (**VLI**) only makes the phase boundaries more natural but does not gain performance improvement. For example, although the SPM method applies the VLI, the dominant functional simulation time is not reduced. In our coarse-grained approach, it does not lead to the reduction of the number of detailed simulated instructions.

While only applying the first-level sampling, more simulation time will be needed for *gcc* than that of 10M SimPoint. The reason is that *gcc* is a complex benchmark [3] [5]. There are 56 iterations in its outmost loop using the reference input set, and their instruction counts vary significantly. Although we only selected two simulation points for *gcc*, the instruction count of one selected simulation point accounts for 60% of the total number of instructions executed in *gcc*. As a result, the COASTS approach needs to simulate more instructions in detail and the total simulation time is significantly increased.

B. Evaluation for Multi-level Sampling

To evaluate our multi-level sampling framework, we apply 10M SimPoint as the fine-grained method in the framework. The speedup and the deviation are shown in Figure 4 and Table II. As the results shown, fewer instructions are simulated in detail in such a multi-level sampling framework and it can achieve a speedup of 14.04X over 10M SimPoint while maintaining a comparable accuracy. Even for **gcc**, our framework achieves 97% performance of the SimPoint method.

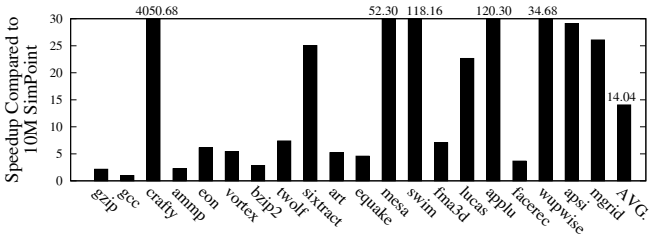

Fig. 4. Speedup of the multi-level sampling over SimPoint

The percentage of detailed simulation instructions for each benchmark is shown in Table III. The results show that fewer instructions are simulated in detail in such a multi-level sampling framework. The underlying reason is that the total instructions in selected coarse-grained samples are much less than that of the entire program. As a result, fewer fine-grained simulation points are needed compared to those used to represent the entire program. Although two-level sampling could lead to a larger accumulation of errors, the increments of error rates are slight shown in Table II. So they are still comparable to those in COASTS and 10M SimPoint. The results in Table II also illustrate the framework is not sensitive to different architectural configurations.

VI. Conclusions

Phase granularity has been one of the most important parameters in representative sampling simulation. However, it has not been well studied yet. Prior researchers simply used a fine-grained strategy in their designs. In this paper, we presented a comprehensive study on the phase granularity of representative sampling simulation. We found that a multi-level sampling strategy can further improve the simulation speed through reducing both the detailed simulation time and the functional simulation time compared to a fine-grained one. Based on the analysis, we designed and implemented a coarse-grained sampling framework (i.e., COASTS) and a multi-level sampling framework. Experimental results showed that both these two systems are effective in simulation speed and accuracy even for the benchmarks with very complex phase behavior, such as *gcc*, *vortex* and *equake* [3][5][9].

Acknowledgment

We thank the anonymous reviewers for their insightful comments. This work was funded by China National Natural Science Foundation under grant numbered 60903015, National 863 Program of China under Grant No. 2012AA010905, Key Project of National 863 Program of China under Grant No. 2009AA012201.

References

[1] J. J. Yi and D. J. Lilja, "Simulation of computer architectures: Simulators, benchmarks, methodologies, and recommendations," *IEEE Transaction on Computers.*, vol. 55, no. 3, pp. 268–280, 2006.

[2] M. J. Hind, V. T. Rajan, and P. F. Sweeney, "Phase shift detection: A problem classification. Technical Report, IBM," 2003.

[3] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *ASPLOS*, 2002.

[4] E. Perelman, G. Hamerly, and B. Calder, "Picking statistically valid and early simulation points," in *PACT*, 2003.

[5] X. Shen, Y. Zhong, and C. Ding, "Locality phase prediction," in *ASPLOS*, 2004, pp. 165–176.

[6] A. S. Dhodapkar and J. E. Smith, "Comparing program phase detection techniques," in *MICRO*, 2003, pp. 217–227.

[7] J. Lau, S. Schoenmackers, and B. Calder, "Structures for phase classification," in *ISPASS*, 2004.

[8] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder, "Motivation for variable length intervals and hierarchical phase behavior," in *ISPASS*, 2005, pp. 135–146.

[9] J. Lau, E. Perelman, and B. Calder, "Selecting software phase markers with code structure analysis," in *CGO*, 2006, pp. 135–146.

[10] R. A. Johnson and D. W. Wichern., *Applied Multivariate Statistical Analysis (5th Edition)*. Prentice Hall, 2002.

[11] M. C. Huang, J. Renau, and J. Torrellas, "Positional adaptation of processors: Application to energy reduction," in *ISCA*, 2003.

[12] D. C. Burger and T. M. Austin, "The simplescalar tool set, version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin - Madison," 1997.

[13] A. A. Nair and L. K. John, "Simulation points for spec 2006," in *ICCD*, 2008, pp. 38–46.