PAPER Special Section on Parallel and Distributed Computing and Networking

An Inductive Method to Select Simulation Points

MinSeong CHOI[†a)], **Takashi FUKUDA**[††], **Masahiro GOSHIMA**[†††], ***Nonmembers***, ***and*** **Shuichi SAKAI**[†], ***Fellow***

SUMMARY The time taken for processor simulation can be drastically reduced by selecting **simulation points**, which are dynamic sections obtained from the simulation result of processors. The overall behavior of the program can be estimated by simulating only these sections. The existing methods to select simulation points, such as **SimPoint**, used for selecting simulation points are deductive and based on the idea that dynamic sections executing the same static section of the program are of the same phase. However, there are counterexamples for this idea. This paper proposes an inductive method, which selects simulation points from the results obtained by pre-simulating several processors with distinctive microarchitectures, based on assumption that sections in which all the distinctive processors have similar istructions per cycle (IPC) values are of the same phase. We evaluated the first 100G instructions of SPEC 2006 programs. Our method achieved an IPC estimation error of approximately 0.1% by simulating approximately 0.05% of the 100G instructions.

key words: *simulation point, sampling simulation, microarchitecture, processor architecture, simulation, computer architecture*

1. Introduction

In modern computer architecture studies, simulation of a target architecture is indispensable to evaluate its performance. However, process or simulation is highly time consuming because of the following two reasons.

Long Simulation Time

First, the execution speed of simulators is low. The ratio of the execution time taken by a simulator to the actual execution time is termed as speed-down (**SD**). In general, the SDs of emulators are approximately 10, whereas the SDs of cycle-accurate simulators are approximately 1000 to 10,000. This is because emulators only reproduce the architectural result of the program instructions, whereas cycle-accurate simulators reproduce cycle-by-cycle behavior of the target processors. When the SD of a simulator is 1000, a 10 min program on a real machine will take 10,000 min to execute, i.e., a week on that simulator.

Second, there are a huge number of instructions in benchmark programs. In the SPEC CPU 2006 [1], which is the most typical benchmark for evaluating the processor performance, the longest program execution consists of as much as 100T instructions, and it will take years to simulate such a program. In addition, it is necessary to simulate each program for tens of combinations of different values of several parameters.

In conclusion, speeding up simulators is an important but not a fundamental solution. Even if several times speedup is achieved, the simulation time will be reduced from years to months. It is still far from practical consideration.

Simulation Points

Therefore, we usually simulate small sections of benchmark programs, typically, the subsequent 100M instructions after the first 1G program instructions, to skip the initialization section from the evaluation result.

Although this method is generally accepted, it is still uncertain whether the simulated sections accurately reflect the characteristics of a program. In fact, the skipping of the first 1G instructions was determined to be insufficient to exclude all initialization sections, such as astar, mcf, and omnetpp in SPEC 2006 [2].

To cope with this problem, the idea of sampling simulation can be considered. Instead of executing the entire program, simulating a set of simulation points, which are the dynamic sections of program execution, is sufficient to estimate the overall behavior of the target processor. Simulating 100M instructions after skipping the first 1G instructions, as stated earlier, can be considered as the simplest but insufficient example of simulation points.

SimPoint

Selection of simulation points is usually performed based on the same idea of **phase detection** [3], [4]. SimPoint [5]–[8], which is the most recognized method, selects simulation points from the program counter (PC) sequence of executed instructions obtained by emulating the target processor. SimPoint assumes that the dynamic intervals of instructions executing the same static section of a program will be of the same phase, and the target processor behaves similarly in these intervals.

Thus, SimPoint categorizes the instruction intervals into clusters based on the PCs included in each interval. If two intervals have almost the same variety of PCs, they are categorized into the same cluster. Then, the representative

Manuscript received January 8, 2016.
Manuscript revised May 9, 2016.
Manuscript publicized August 24, 2016.

[†]The authors are with Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, 113–8656 Japan.
[††]The author is with Tokyo Software & Systems Development Laboratory, IBM Japan, Tokyo, 103–8510 Japan.
[†††]The author is with Information Systems Architecture Research Division, National Institute of Informatics, Tokyo, 101–8430 Japan.
a) E-mail: choi@mtl.t.u-tokyo.ac.jp

DOI: 10.1587/transinf.2016PAP0030

interval from each of the clusters is selected as one of the simulation points. The IPC (or other performance metrics) of the program will be extrapolated as an average of simulation results of the selected simulation points weighted by the number of intervals in each cluster.

Proposal

However, there are counterexamples about the assumption that the target processor executing the same static section of the program behaves similarly. Even when the target processor is executing the same loop or function, the cache hit rates will differ depending on the amount of data it refers, and the IPC can be drastically different. In reality, as detailed in Sect. 4, the IPC of the 483.xalancbmk program of SPEC 2006 [1] gradually degrades because the working set size gradually increases even executing the same static intervals. Thus, this paper proposes a method to select simulation points from the simulation results of several processors with distinctive microarchitectures. SimPoint can be referred to as deductive, whereas the proposed method can be referred to as **inductive**.

The remainder of this paper is organized as follows: Section 2 describes SimPoint and summarizes its limitations. The proposed method is explained in Sect. 3, which begins with an analogy of a racing circuit, and followed by the detailed description of the method's procedure. Section 4 shows the evaluation results. The related literature excepting SimPoint is summarized in Sect. 5.

2. SimPoint

In this section, we describe SimPoint [5]–[8] and summarize its limitations.

2.1 SimPoint

As previously described, SimPoint selects simulation points based on the assumption that the dynamic sections executing the same static section of a program are of the same phase.

Pre-Emulation

First, SimPoint obtains the sequence of executed instructions by pre-emulating the target program. This sequence is divided into **intervals** of a fixed length. The interval length is typically 1M to 100M instructions.

Basic Block Vector

SimPoint categorizes these intervals into clusters according to the **basic block vector** of each interval. Each element of the basic block vector denotes the number of executions of each basic block in the interval. The number of dimensions of basic block vector is given by the number of basic blocks executed more than once throughout the execution of the target program. However, in an interval, only hundreds or thousands of basic blocks are executed resulting in hundreds or thousands of non-zero elements in the basic block vector. Therefore, basic block vectors will be high-dimensional sparse vectors.

Clustering, Selection, and Estimation

SimPoint clusters intervals by using the ***k*-means** method [9], which is suitable to cluster sparse vectors such as basic block vectors.

Finally, the nearest interval to the weighted center of each cluster is selected as one of the simulation points.

To estimate the IPC or other performance metrics, the target processor only simulates the selected simulation points of the program. The metrics is then deduced as an average of the simulation results of the simulation points weighted by the number of intervals in each cluster.

2.2 Limitation of SimPoint

As previously described, there are counterexamples about the assumption that the target processor executing the same static sections of the program behaves similarly. Even when the target processor is executing the same loop or function, the cache hit rate is different depending on the amount of data it refers, and the IPCs can be drastically different. Thus, the accuracy of SimPoint degrades in some benchmark programs, as shown in Sect. 4.

The cache hit rate is affected by the cache size, which is the microarchitectural parameter of the target processor. Therefore, to overcome this limitation, information about the microarchitecture must be considered. However, SimPoint only uses the information about the target program. One of the most obvious reasons is that even if simulation points are selected for the specific microarchitecture, it is uncertain whether the simulation points can be used for another microarchitecture.

This paper proposes an **inductive** method, which selects simulation points that can be used for general microarchitectures from the simulation results of several processors with distinctive microarchitecture.

3. Inductive Method

This section describes the proposed **inductive** method for selecting simulation points. Section 3.1 introduces an analogy for a better understanding of the proposed method, and Sect. 3.2 describes the procedure of the method.

3.1 Analogy of Racing Circuit

The following analogy of a racing circuit is helpful in understanding the proposed method. Assume that we want to estimate the runtime of a very long circuit for an unknown transportation model X without making it run the entire length of the circuit. Our method is as follows.

First, in advance, we make several models run the entire length of the circuit to measure the speed of each model in all the sections of the circuit. Figure 1 shows the results for a racing car, a scooter, and a walker. For example, the speed of the racing car increases in the second half of the

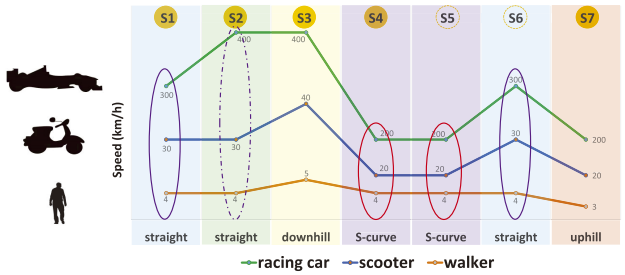


Fig. 1 Speeds of three transportation models in sections of a circuit (km/h).

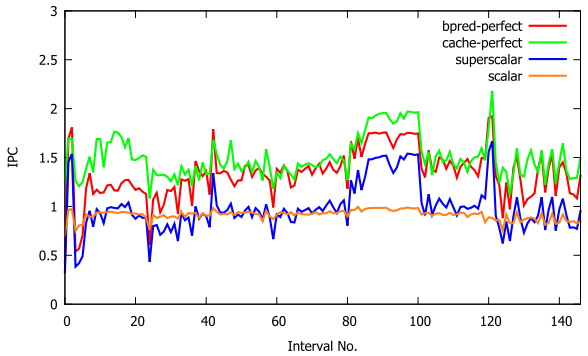


Fig. 2 IPC transitions of four models for 400.perlbench with *test* input.

long straight section (S2) and downhill (S3), and decreases in the S-curves (S4 and S5) and uphill (S7). The speeds of the other models also change according to their characteristics.

Although we denote the type of sections as straight, S-curve, uphill, and downhill to illustrate the cause of speed change, our method does not determine what the sections really are. All that is needed in our method is the resulting speed values.

In both S1 and S6, the speeds of all the three models are $(300, 30, 4)$. Therefore, S1 and S6 can be considered as similar sections. The same applies to S4 and S5.

Next, to estimate the runtime of an unknown transportation model X, we do not need to run it in S5 and S6, instead, we can use the speeds in S4 and S1, respectively. The runtime of model X can be estimated by running it in sections marked by a yellow solid circle in Fig. 1. These sections play the role of simulation points in the sampling simulation.

For this method to work, using the transportation models beforhand is essentially important. They must be sufficiently distinctive from each other. For example, if the walking speed is not used, S7 can be regarded as similar to S4 and S5. As a result, the estimation accuracy for model X possibly degrades.

Processor Simulation

The same method can be applied to processor simulation. If the results for the full-length execution are available for different processors with distinctive microarchitectures, we can select a set of simulation points by comparing the IPC values in this case.

Figure 2 shows the IPC transitions of four distinctive models for full-length execution of 400.perlbench with *test* input. In this figure, we can observe that the phases are conserved for the different models. Furthermore, an unknown model is expected to have a similar IPC transition.

Difference with SimPoint

By considering the analogy of a racing circuit, SimPoint and other methods based on PCs are regarded as methods to detect the phases of the circuit from the static characteristics of the sections. That is, in Fig. 1, they will categorize two straight sections S1 and S2 into the same phase, for example, by comparing the geometric shapes of these sections. However, the speed of the racing car is different in S1 and S2, and it is possible that the estimation accuracy for model X will degrade.

This is because the speed in a section changes depending on the dynamic state when a vehicle is approaching the section. A racing car can run faster on long straight sections. Similarly, a modern processor can run faster if a loop is longer, because branch prediction and/or the cache hit rate become higher.

In contrast, our method does not consider the static characteristics of the sections but considers the resulting speed, which is affected by both the static characteristics of the sections and the dynamic state of the vehicle.

3.2 Pre-Simulation by Using Basis Models

As described in the previous sections, SimPoint selects simulation points based on a PC sequence obtained by pre-emulating benchmark programs. In contrast, our method detects simulation points based on an IPC sequence obtained by pre-simulating benchmark programs with distinctive models.

The proposed method first obtains IPC values for each interval by pre-simulating benchmark programs with several models. We refer to these models as **basis models**. As the previously mentioned circuit analogy suggests, the basis models should be distinctive from each other. The method to choose good basis models is explained in detail in Sect. 3.6.

3.3 IPC Vector

By pre-simulating two intervals with n basis models, two n-dimensional **IPC vectors** $v_1 = (i_{1_1}, i_{1_2}, \ldots, i_{1_n})$ and $v_2 = (i_{2_1}, i_{2_2}, \ldots, i_{2_n})$ are obtained.

If the distance between v_1 and v_2 is sufficiently small, i.e., $i_{1_1} \simeq i_{2_1}$, $i_{1_2} \simeq i_{2_2}, \ldots, i_{1_n} \simeq i_{2_n}$, these two intervals show similar IPC values for all the basis models. Therefore, two intervals can be expected to be of the same phase, and for an unknown model m_{n+1}, $i_{1_{n+1}} \simeq i_{2_{n+1}}$ can be expected, and obtaining only $i_{1_{n+1}}$ with simulation is sufficient to estimate $i_{2_{n+1}}$. Therefore, our method clusters all the IPC vectors based on the distance between them. Similar to SimPoint,

one simulation point is then selected from each cluster.

Note that IPC vectors are low-dimensional dense vectors. The number of dimensions of IPC vectors is small (i.e. four, as shown in Sect. 4) because it is dependent on the number of basis models. That is, the IPC vectors in our method are low-dimensional (dense) vectors. In contrast, the basic block vectors of SimPoint are high-dimensional sparse vectors as explained in Sect.2.

3.4 Clustering IPC Vectors

Unlike SimPoint, which uses the k-means method, our method uses the following simple algorithm to cluster IPC vectors:

1. For a new IPC vector v, the distances to all the clusters are calculated. The Euclidean distance to the **gravity center** of a cluster is typically used for these distances.
2. If the distance from v to any cluster is less than a given threshold, v is appended to the nearest cluster.
3. Otherwise, a new cluster is created, and v is appended to that cluster as the first member.

The clustering threshold in this algorithm affects the tradeoff between the amount of simulation points and the accuracy of the estimation. The threshold gives the maximum *radius* of each cluster in the vector space. Thus, a smaller threshold results in a larger number of smaller clusters; and then, a larger number of simulation points and a higher accuracy of the estimation.

In this manner, the number of clusters is automatically decided by the threshold value, whereas determining the optimal k value of the k-means method is difficult [9].

This simple algorithm works because IPC vectors are low-dimensional dense vectors. In contract, SimPoint requires complex algorithms, such as k-means, because basic block vectors are high-dimensional sparse vectors.

3.5 Fixed-Length Interval

Similar to SimPoint, the proposed method divides the program execution into sections. We used fixed-length intervals, as used in SimPoint.

Fixed-length intervals can degrade the estimation accuracy because a single phase can have phase changes [10]–[12].

However, our proposed method can mitigate this problem by using shorter intervals. Our method clusters the vectors more efficiently than SimPoint even with the increased number of shorter intervals, because IPC vectors are low-dimensional dense vectors, whereas basic block vectors are high-dimensional sparse vectors. In other words, the proposed method can adopt finer intervals to lower the impact of phase changes.

3.6 Basis Models

As the analogy in Sect. 3.1 suggests, the basis models should be sufficiently distinctive from each other.

We selected the basis models based on the IPC degradation factors of processors. The following are the three main IPC degradation factors of modern superscalar processors:

1. cache miss
2. branch prediction miss
3. dependency among instructions

In general, studies on processors focus on increasing or maintaining IPC by decreasing the effect of these factors.

In Sect. 4, we use the following four basis models for evaluation:

superscalar A moderate 4-way out-of-order superscalar processor. Table 2 summarizes the configuration.
scalar A scalar processor with a cache.
cache-perfect A superscalar processor with a perfect cache
bpred-perfect A superscalar processor with a perfect branch predictor.

Table 1 summarizes these basis models, and displays whether the three IPC degradation factors have an influence on them.

The basis models other than superscalar are not affected by one or two of the three IPC degradation factors. Thus, these models can be considered as the lower limits in IPC degradation, and the IPC degradation of all realistic processors is larger than those. Therefore, these models play the role of *basis vectors* to represent realistic processors.

Table 1 Basis Models & IPC Degradation Factors.

Basis Model	1) cache miss	2) bpred miss	3) inst. dep.
superscalar	✓	✓	✓
scalar	✓		
cache-perfect		✓	✓
bpred-perfect	✓		✓

Table 2 Configuration of superscalar Model.

ISA	Alpha w/ byte/word ext.
pipeline stages	Fetch:3, Rename:2, Dispatch:2, Issue:4
fetch width	4 inst.
issue width	Int:2, FP:2, Mem:2
inst. window	Int:32, FP:16, Mem:16
branch pred.	8KB g-share
BTB	2K entries, 4way
RAS	8 entries
L1C	32KB, 4way, 3cycles, 64B/line
L2C	4MB, 8way, 15cycles, 128B/line
main memory	200cycles

4. Evaluation

In this section, we provide the evaluation and comparison of our method with SimPoint. Section 4.1 summarizes the methodology and Sects. 4.2 and 4.3 display the results.

Table 3 Averaged relative IPC of target models to superscalar.

Target Model	Relative IPC
cache-half	−7.6%
pht-single	−2.5%
eight-way	11.1%
regcache	−9.2%

4.1 Evaluation Methodology

Simulator

We used the fully cycle-accurate Onikiri 2 simulator [13], that is, it reproduces the behavior of instructions in each stage in the accurate cycles. It executes instructions in the accurate execute stages, and verifies the results with those of an on-line emulator in the commit stage. Thus, the behavior after mispredictions is also accurately reproduced.

The Onikiri 2 was used to evaluate several advanced studies on microarchitecture [14]–[16], including a register cache system [14] that we evaluated in this paper.

Benchmark Programs

We used all of the 22 programs of SPEC CPU 2006 with *ref* input [1].

Although we could only use the first 100G instructions of each program owing to time constraints. This number is three orders of magnitude greater than the instructions evaluated in the existing studies on microarchitecture [14]–[16].

We selected simulation points from the 100G instructions, estimated IPC values by using SimPoint and our method, and calculated the estimation errors. True IPC values were obtained by simulating the 100G instructions.

Basis Models

We used the four basis models mentioned in Sect. 3.6.

Target Models

The IPCs of the following four target models were estimated using the simulation points obtained through SimPoint and our method:

cache-half A superscalar processor in which the numbers of ways of the caches are half-sized.

pht-single A superscalar in which the bit width in the PHT of the branch predictor decreased from 2 to 1.

eight-way An 8-way out-of-order superscalar processor after the Intel Haswell [17] and IBM POWER8 [18] processors.

regcache A superscalar processor with a register cache.

Table 3 summarizes and compares the relative IPC of the target models with those of the superscalar model. The first three models are simple variations of the basis models, and show how our method reflects the IPC degradation factors. In contrast, the regcache model, detailed as follows, is more realistic to estimate.

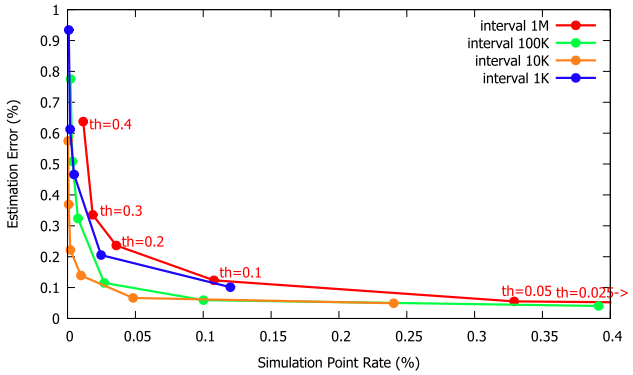


Fig. 3 Estimation error against simulation point rate of our method.

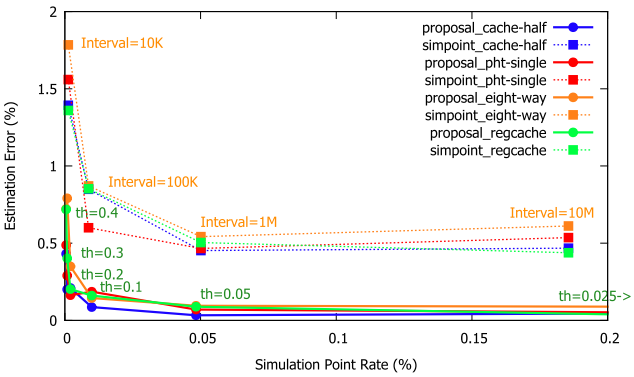


Fig. 4 Estimation error against simulation point rate of SimPoint and our method.

The regcache Model

The regcache model adopts the Non-latency Oriented Register Cache System proposed by Shioya et al. [14]. This register cache system causes one-cycle stall in the backend pipeline when more than two (number of register file ports) register cache misses occur in a single cycle.

When the register cache is sufficiently large (e.g., eight entries), this system has slight IPC degradation, that is, the regcache model shows almost the same IPC as the superscalar model. Therefore, we set the register cache size to only 2 entries for a sensible evaluation. As a result, the IPC of regcache model degraded by 9.2% of the superscalar model (Table 3).

The IPC degradation factor of the regcache model is register cache miss, which is different from the three IPC degradation factors described in Sect. 3.6. Thus, the result of this model is the most important because it will verify if our method works for target models that have general IPC degradation factors other than the three mentioned earlier.

4.2 Evaluation Results

In this section, we describe the use of graphs Figs. 3 and 4 to show the estimation error against the amount of simulation points.

In these graphs, the x-axis shows the **simulation point**

rate, i.e., the rate of number of instructions in the selected simulation points to the number of all the instructions in a benchmark program. Thus, the number of instructions is represented by the simulation point rate, and the number of simulation points is calculated by the simulation point rate and the length of the intervals.

The y-axis shows the **estimation error**, i.e., the rate of estimated IPC obtained by simulating only the selected simulation points to the *true* IPC obtained by simulating all the instructions in a benchmark program.

In general, because there is a tradeoff between the simulation point rate and the estimation error, negatively sloped curves are plotted. The nearer the curve is to the origin, the better simulation points are selected. That is, a more precise IPC estimation is achieved by less simulated instructions.

Interval Length of Our Method

SimPoint has only one parameter, i.e., the interval length, whereas our method has two parameters, i.e., the interval length and the clustering threshold. Thus, we first set the interval length of our method.

Figure 3 shows the estimation error against the simulation point rate of our method for different interval lengths. In this figure, four curves are plotted for four interval lengths of 1K, 10K, 100K, and 1M instructions, and each curve is plotted for six thresholds of 0.025, 0.05, 0.1, 0.2, 0.3, and 0.4. For each interval length and threshold, we have 88 pairs of the estimation errors and simulation point rates for 22 benchmark programs of each of the four target models. Each point on the curves is plotted for a pair of the geometric mean of the 88 estimation errors and the geometric mean of the 88 simulation point rates. The point for an interval length of 1K instructions and a threshold of 0.025 is not plotted because the clustering program did not finish even in three days for some pairs out of 88 owing to a huge number of clusters.

In the graph, the curve for 10K instructions is the nearest to the origins of all clustering thresholds, indicating that 10K is optimal. Hereafter, the interval length is fixed to 10K instructions for our method unless otherwise stated.

Overall Results

Figure 4 has two groups of curves. The group of dashed curves represents the result of SimPoint, and the group of solid curves represents those of our method. Each group has four curves for the four target models, and each curve is plotted for the geometric means of 22 estimation errors and 22 simulation point rates for 22 benchmark programs. Note that the four markers of the same parameter are vertically arranged, because the simulation point rates do not vary for the different target models.

As previously described, SimPoint has only one parameter, whereas our method has two. The curves for SimPoint are plotted by changing the interval lengths: 10K, 100K, 1M, and 10M instructions. In contrast, the curves for our method are plotted by changing the clustering thresholds: 0.025, 0.05, 0.1, 0.2, 0.3, and 0.4, with the fixed interval length to 10K instructions. (10K was found to be optimal by Fig. 3). That is, the curve for 10K-instruction interval in Fig. 3 is the geometric mean of the four curves for the four target models used in our method in Fig. 4.

Figure 4 shows the four curves in each group crowding together, indicating that the estimation errors do not widely vary for the different target models.

The group of curves of our method is nearer to the origin than SimPoint, implying that our method can select better simulation points. If the rate of simulation points is approximately 0.05%, the error can be improved from approximately 0.5% to 0.1%. If approximately 0.5% of errors of SimPoint can be acceptable, the rate of simulation points can be reduced to less than 0.01%. That is, a 0.5% error rate can be achieved by simulating only $100\text{G} \times 0.01\% = 10\text{M}$ out of 100G instructions.

In addition, the error of SimPoint is saturated at larger rates of simulation points. As the interval length decreases, the error increases on the contrary. The errors of SimPoint is minimized at the point of 1M instructions [5]–[8].

Our method can more freely select any tradeoff points between the rate of simulation points and the estimation error than SimPoint by changing the clustering threshold.

Interval Lengths

SimPoint needs longer intervals than 1M instructions would comes from the limitation of its clustering ability. It is essentially difficult to cluster a huge number of sparse vectors.

In contrast, our proposal does not work well for 1K-instruction intervals. The reason would be that 1K instructions is essentially too short to stably measure the IPC. For example, because the latency of the main memory is hundreds of cycles, the IPC of 1K-instruction intervals varies by tens of percent by a coincidental last-level cache miss.

Note that there is no way to prove this speculation. If the IPC of one out of two intervals is degraded only by a coincidental cache miss, they should be regarded as different phases by definition. In other words, a phase loses its meaning for short intervals such as 1K instructions.

Selected Intervals as Simulation Points

Figure 5 shows simulation points for 483.xalancbmk se-

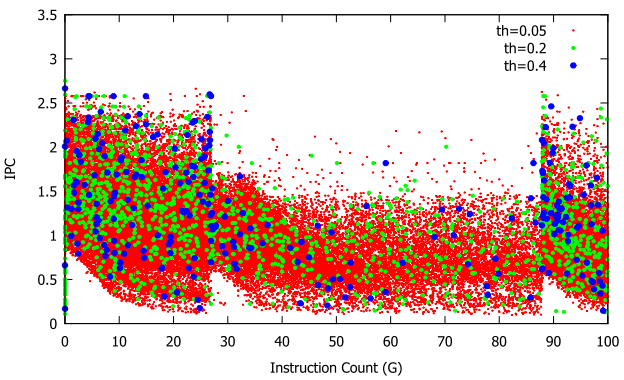


Fig. 5 Selected simulation points for 483.xalancbmk.

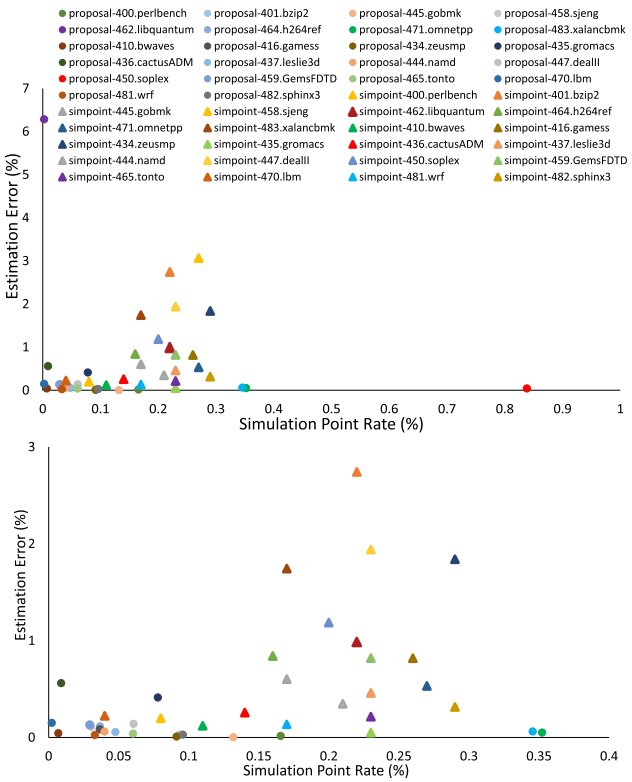


Fig. 6 Estimation error against simulation point rate of regcache.

lected by our method. In this graph, the x-axis shows the dynamic interval number measured in instructions from 0 to 100G, and the y-axis shows IPC. Each point is plotted for an interval selected as a simulation point and its IPC on the superscalar model. There are three groups of points plotted for three thresholds: 0.05, 0.2, and 0.4.

For these three thresholds, the numbers of selected 10K-instruction intervals are 34,546, 884, and 154; and the geometric means of the estimation errors for the four target models are 0.0715%, 0.179%, 1.22%, respectively.

The variation in IPC is wider than expected or than Fig. 8 in Sect. 4.3. This is because of the shorter interval length, 10K instructions in this figure and 1M instructions in Fig. 8. Higher or lower IPC values observed in shorter intervals are averaged in a longer interval.

Per-Program Results of regcache Model

Figure 6 and Table 4 show the estimation errors against simulation point rates of each of the 22 benchmark programs on the regcache model. In Fig. 6, the triangles represent the plots for SimPoint, whereas the circles represent the plots for our method. The lower graph in the figure is an enlarged view of the upper graph. The interval lengths of SimPoint and our method are 10M and 10K instructions respectively, and are the optimal values for each method as obtained from the previous results. The clustering threshold of our method is 0.05, which is selected for better precision than the rate of simulation points.

Obviously, the plots of our method are concentrated around the origin, whereas those of SimPoint are father from the origin. This indicates that our method outperforms SimPoint for most of the programs. The graph shows that our method achieves both lower rates of simulation points and lower error rates 16 out of the 22 programs.

Table 4 Estimation error (EE) against simulation point rate (SPR) of regcache.

program	Proposal		SimPoint	
	SPR (%)	EE (%)	SPR (%)	EE (%)
400.perlbench	0.16561	0.01694	0.27000	3.06510
401.bzip2	0.09348	0.02843	0.22000	2.74195
445.gobmk	0.13175	0.00298	0.17000	0.60204
458.sjeng	0.03027	0.10885	0.08000	0.19820
462.libquantum	0.00221	6.28601	0.22000	0.98451
464.h264ref	0.03662	0.11553	0.16000	0.84040
471.omnetpp	0.35213	0.05085	0.27000	0.53224
483.xalancbmk	0.34546	0.06200	0.17000	1.74393
410.bwaves	0.00683	0.04421	0.11000	0.12033
416.gamess	0.03645	0.08479	0.26000	0.81849
434.zeusmp	0.09130	0.01052	0.29000	1.84017
435.gromacs	0.07791	0.41340	0.23000	0.04990
436.cactusADM	0.00889	0.56141	0.14000	0.25654
437.leslie3d	0.04766	0.05636	0.23000	0.45934
444.namd	0.03986	0.06121	0.21000	0.34736
447.dealII	0.06057	0.14103	0.23000	1.94106
450.soplex	0.83813	0.04425	0.20000	1.18586
459.GemsFDTD	0.02940	0.12953	0.23000	0.81928
465.tonto	0.06017	0.04078	0.23000	0.21437
470.lbm	0.00219	0.15174	0.04000	0.22357
481.wrf	0.03287	0.02536	0.17000	0.13511
482.sphinx3	0.09580	0.03031	0.29000	0.31499
GEOMEAN	0.04832	0.07033	0.18557	0.53591

In the four cases of 462.libquantum on regcache, 471.omnetpp and 444.namd on cache-half, and 465.tonto on eight-way, SimPoint outperforms our method in some parameters. The reason for this will be partly discussed in the following subsection through a few examinations.

As a whole, our method also works well for regcache model, which has a IPC degradation factor different from the three described in Sect. 3.6.

This is partly because the IPC degradation of regcache model is caused by the number of register accesses per cycle, it is strongly correlated to the instruction-level parallelism of each interval, which can be distinguished by the difference in IPC between the scalar and superscalar models.

Therefore, if a target model has another IPC degradation factor that is not correlated to the instruction-level parallelism, our method will need another basis model to distinguish that factor.

4.3 Detailed Results

This section visualizes the phases of some characteristic results. In this section, we denoted a single phase as a point in graphs by using an interval length of 1M instructions for both methods.

401.bzip2 on regcache

401.bzip2 is one of the benchmark programs that showed the best results through our method.

Figure 7 shows the IPC of clustered intervals for 401.bzip2 on the regcache model. The upper and the

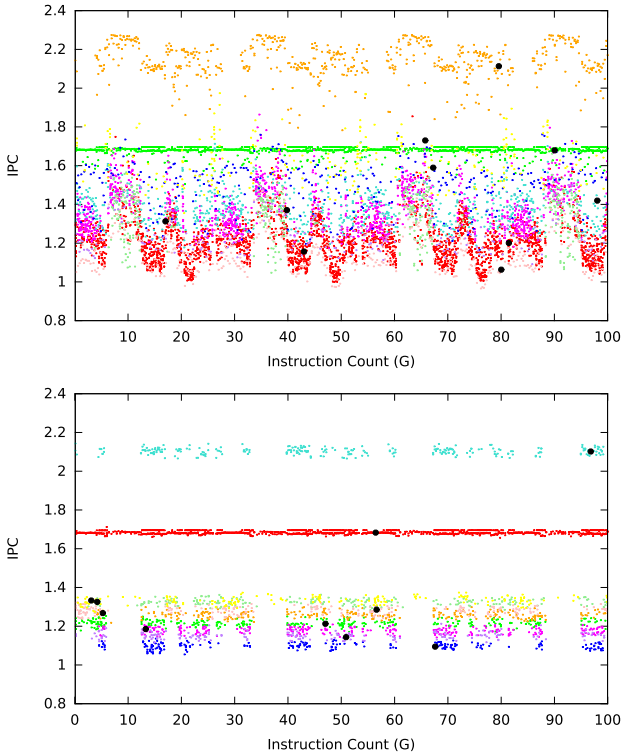


Fig. 7 IPCs of intervals in top ten clusters for bzip2 on regcache by using SimPoint (upper) and proposal (lower).

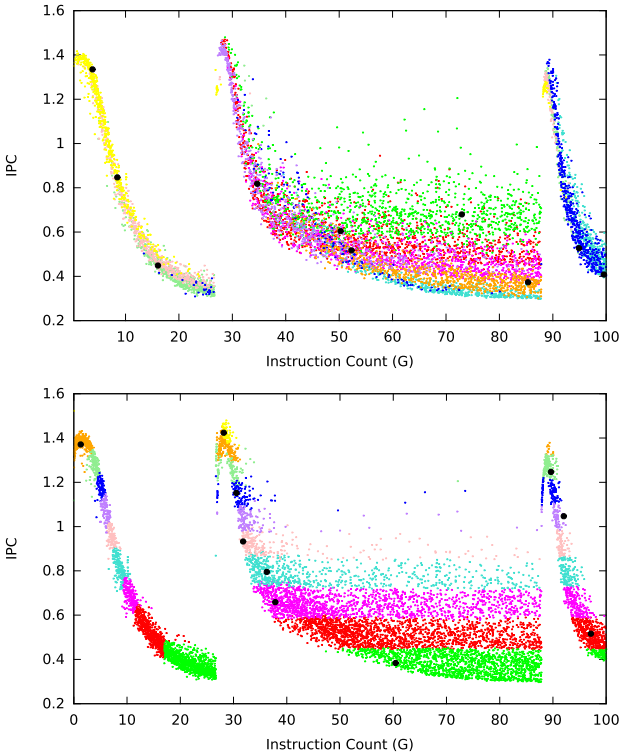


Fig. 8 IPC of intervals in top ten clusters for xalancbmk on eight-way by using SimPoint (upper) and proposal (lower).

lower graphs of Fig. 7 show the results of SimPoint and our method, respectively. As in Fig. 5, the x-axis represents the interval number in instructions, and the y-axis represents the IPC.

The intervals clustered to the same cluster are painted with the same color. Because the graphs will be filled with colors if all intervals are plotted, only the largest 10 clusters are plotted. In each graph, the intervals in the clusters are painted with different colors (gnuplot default colors) in the descending order of cluster size. Thus, the occurrence of colors in two graphs has no relevance. Each of the black points is the selected intervals as a simulation point from each cluster.

In the lower graph of Fig. 7, our method shows an ideal result. A straight horizontal stripe of different colors in the graph show that the intervals of almost identical IPC are clustered to the same cluster.

In contrast, in the upper graph, different colors are mixed together, indicating that the IPC of the same color, i.e., the same cluster widely vary in SimPoint. As a result, a selected interval as the simulation point from a cluster does not represent the other intervals in that cluster in regard to IPC, resulting in a poor IPC estimation.

483.xalancbmk on eight-way

As previously described, 483.xalancbmk executes the same loop with increasing working set. As a result, the cache hit rate and the resulting IPC are gradually decreased.

Figure 8 shows the IPC of the intervals in the top ten

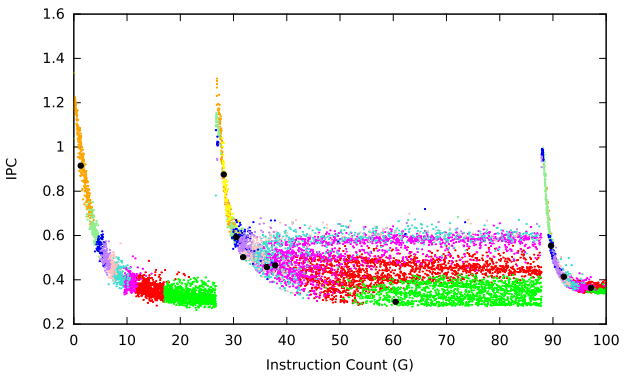


Fig. 9 IPC of intervals in top ten clusters for xalancbmk on cache-half. Proposal.

cluster for 483.xalancbmk on the eight-way model. The upper and lower graphs show the results of SimPoint and our method. Although horizontal stripes of different colors are observed in both the graphs, the stripe of SimPoint is curved while the stripe of our method is not.

The stripe of SimPoint shows that the same set of static sections are repeatedly executed, because SimPoint clusters intervals based on the PC. Then, the curve of the stripe shows that the IPC of these static sections gradually degraded at the same rate as one another, because of the gradually increasing working set.

In contrast, in Fig. 7, the straight stripe of our method shows that intervals of almost identical IPC, irrespective of

the PC, are clustered to the same cluster.

483.xalancbmk on cache-half

Nevertheless, for 483.xalancbmk on the cache-half model in Fig. 9, the graph of our method does not show horizontal stripes as shown in the lower graphs of Figs. 7 and 8, which indicates that our method also cannot detect good phases for different sizes of caches.

5. Related Work

This section introduces related studies such as software phase marker, SimFlex, and Live-Points.

Software Phase Marker

Lau et al. proposed an automated profiling approach to identify code locations whose executions correlate with phase changes [10]. They called these code constructs **software phase markers**, which are used to detect phase changes across different inputs to a program without hardware support.

They built a hierarchical call-loop graph, which is an extended call graph with nodes for loops to represent the flows between the functions and loops in a program. The edge of the graph tracks the maximum, average, and standard deviation in hierarchical instruction counts on the edge path. A software phase marker is instrumented at the edge of low standard deviation as a fraction of the average of hierarchical instruction counts. The phase markers can also be used to create variable length intervals to guide SimPoint. A new variable length interval starts when a new phase marker is observed during execution. However, the evaluation results showed lower accuracy for some benchmark programs than that obtained through the original SimPoint, because the makers tended to produce extremely short intervals for IPC estimation. The authors claimed that the benefit for SimPoint is not an improvement of accuracy or reduction in simulation time;however, a new feature must be provided in which simulation points can be mapped to the source code so that the simulation points can be re-used if the program is recompiled, even on a different instruction sets.

SimFlex and Live-Points

Hardavellas et al. proposed **SimFlex** [19]–[22], which randomly selects simulation points for sampling simulation. They introduce the confidence interval for an acceptable margin of error on a statistical basis.

Although G. Hamcrly et al. [6] claimed that SimPoint is better than random sampling, Hardavellas et al. argued that the merit of SimFlex is that it can provide statistical reliability to users.

The group of SimFlex also proposed **Live-Points**, which checkpoints the cache and the branch predictors to avoid frequent cold starts in sampling simulation [23].

6. Conclusion

This paper proposed an **inductive** method to select simulation points. This method inductively selects simulation points from the simulation results of several processors with distinctive microarchitecture.

We selected simulation points of SPEC CPU 2006 benchmark programs, and estimated the IPC of several target architectures including advanced microarchitectures from recent studies. The evaluation results show that our method achieves more accurate estimation with less simulation points than SimPoint.

However, our method is still insufficient for different sizes of caches. Thus, we are currently evaluating a basis model with vast levels of caches, which will show different IPC for slightly different working set sizes.

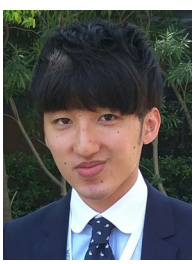
In addition, we plan to evaluate an inductive method without the superscalar model. Although the IPC sequences of the other basis models can be obtained by emulation, the superscalar model needs simulation, which takes 10 to 100 times more time. If a method without simulation can be established, the phase detecting time will be drastically reduced.

References

[1] SPEC CPU 2006. http://www.spec.org/cpu2006/
[2] A.A. Nair and L.K. John, "Simulation points for SPEC CPU 2006," Proc. Int'l Conf. on Computer Design (ICCD), pp.397–403, Oct. 2008.
[3] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder, "Discovering and exploiting program phases," IEEE Micro, vol.23, no.6, pp.84–93, 2003.
[4] M. Hind, V. Rajan, and P.F. Sweeney, "Phase detection: A problem classication," Tech. Rep., IBM Research, 2003.
[5] E. Perelman, G. Hamerly, and B. Calder, "Picking statistically valid and early simulation points," Proc. Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), pp.244–255, Sept. 2003.
[6] G. Hamerly, E. Perelman, and B. Calder, "How to use SimPoint to pick simulation points," ACM SIGMETRICS Performance Evaluation Review, vol.31, no.4, pp.25–30, March 2004.
[7] G. Hamerly, E. Perelman, J. Lau, and B. Calde, "SimPoint 3.0: Faster and more flexible program analysis," J. Instruction-Level Parallelism (JILP), vol.7, no.4, pp.1–28, Sept. 2005.
[8] G. Hamerly, E. Perelman, J. Lau, B. Calder, and T. Sherwood, "Using machine learning to guide architecture simulation," J. Machine Learning Research (JMLR), vol.7, no.2, pp.343–378, Feb. 2006.
[9] G. Hamerly and C. Elkan, "Learning the k in k-means," Tech. Rep. CS2002-0716, University of California, May 2002.
[10] J. Lau, E. Perelman, and B. Calder, "Selecting software phase markers with code structure analysis," Proc. Int'l Symp. on Code Generation and Optimization (CGO), pp.135–146, 2006.
[11] Y. Akamatsu, M. Goshima, and S. Sakai, "A phase detection technique not using fixed-length intervals," Proc. Symp. on Advanced Computing Systems and Infrastructures (SACSIS), pp.157–165, May 2011. (in Japanese).
[12] K. Hayakawa, N. Kurata, and S. Sakai, "A phase detection technique with variable-length segments," IPSJ SIG Report 2013-ARC-206 (SWoPP), no.26, pp.1–9, Aug. 2013. (in Japanese).
[13] "Onikiri 2." https://github.com/onikiri/onikiri2/

[14] R. Shioya, K. Horio, M. Goshima, and S. Sakai, "Register cache system not for latency reduction purpose," Proc. Int'l Symp. on Microarchitecture (MICRO), pp.301–312, Dec. 2010.
[15] R. Shioya and H. Ando, "Energy efficiency improvement of renamed trace cache through the reduction of dependent path length," Proc. 32nd IEEE Int'l Conf. on Computer Design (ICCD), pp.416–423, Oct. 2014.
[16] R. Shioya, M. Goshima, and H. Ando, "A front-end execution architecture for high energy efficiency," Proc. 47th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO), pp.419–431, Dec. 2014.
[17] K. Krewell, "Intel's Haswell cuts core power," Microprocessor Report, Sept. 2012.
[18] B. Sinharoy, J.A. Van Norstrand, R.J. Eickemeyer, H.Q. Le, J. Leenstra, D.Q. Nguyen, B. Konigsburg, K. Ward, M.D. Brown, J.E. Moreira, D. Levitan, S. Tung, D. Hrusecky, J.W. Bishop, M. Gschwind, M. Boersma, M. Kroener, M. Kaltenbach, T. Karkhanis, and K.M. Fernsler, "IBM POWER8 processor core microarchitecture," IBM J. Research and Development, vol.59, no.1, pp.2:1–2:21, 2015.
[19] N. Hardavellas, S. Somogyi, T.F. Wenisch, R.E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J.C. Hoe, and A.G. Nowatzyk, "SimFlex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," ACM SIGMETRICS Performance Evaluation Review, vol.31, no.4, pp.31–34, March 2004.
[20] T.F. Wenisch and R.E. Wunderlich, "SimFlex: Fast, accurate and flexible simulation of computer systems," Tutorial in the Int'l Symp. on Microarchitecture (MICRO), Nov. 2005.
[21] T.F. Wenisch, R.E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsaf, and J.C. Hoe, "SimFlex: Statistical sampling of computer system simulation," IEEE Micro, special issue on Computer Architecture Simulation, vol.26, no.4, pp.18–31, Aug. 2006.
[22] "SimFlex: Fast, accurate & flexible computer architecture simulation." http://parsa.epfl.ch/simflex/
[23] T.F. Wenisch, R.E. Wunderlich, B. Falsafi, and J.C. Hoe, "Simulation sampling with live-points," Proc. Int'l Symp. on Performance Analysis of Systems and Software (ISPASS), pp.2–12, March 2006.

MinSeong Choi was born in 1977. She is currently a doctoral student in Graduate School of Information Science and Technology, The University of Tokyo. She received her M.E. degree in Department of Electrical and Computer Engineering from Hanyang University in 2005. From 2005 to 2010, she was an engineer at Hynix, Inc.



Takashi Fukuda was born in 1990. He is currently an engineer work at IBM Japan. He received his M.E. degree in Information and Communication Engineering from The University of Tokyo in 2015. He was awarded the Young Scientist Award of The IPSJ Special Interest Group on System Architectrue in 2014.

Masahiro Goshima was born in 1968. He received his M.E. in engineering and Ph.D. in informatics from Kyoto University in 1994 and 2004, respectively. He was a research fellow of the Japan Society for the Promotion of Science from 1994. From 1996, he was an assistant professor in Kyoto University. From 2005, he was an associate professor in the University of Tokyo. Since 2014, he has been a professor in National Institute of Informatics. He has been engaging in the research area of computer architecture. He received IPSJ Yamashita SIG research award and IPSJ best paper award in 2001 and 2002, respectively. He wrote a book titled "Digital Circuits". He is a member of IPSJ and IEEE.

Shuichi Sakai was born in 1958. He received his M.E. and Ph.D. degree in Information and Communication Engineering from The University of Tokyo in 1983 and 1986, respectively. He was a associate professor at University of Tsukuba in 1996. Since 1997, he is an assistant professor/professor at the Graduate School of Information Science and Technology, the University of Tokyo. He is a member of IPSJ, IEICE and IEEE.