2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip

Accelerating Multicore Architecture Simulation Using Application Profile

Keiji Kimura, Gakuho Taguchi, Hironori Kasahara
Department of Computer Science and Engineering, Waseda University
3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan
Email: keiji@waseda.jp, gakuho@kasahara.cs.waseda.ac.jp, kasahara@waseda.jp

***Abstract*—Architecture simulators play an important role in exploring frontiers in the early stages of the architecture design. However, the execution time of simulators increases with an increase the number of cores. The sampling simulation technique that was originally proposed to simulate single-core processors is a promising approach to reduce simulation time. Two main hurdles for multi/many-core are preparing sampling points and thread skewing at functional simulation time. This paper proposes a very simple and low-error sampling-based acceleration technique for multi/many-core simulators. For a parallelized application, an iteration of a large loop including a parallelizable program part, is defined as a sampling unit. We apply X-means method to a profile result of the collection of iterations derived from a real machine to form clusters of those iterations. Multiple iterations are exploited as sampling points from these clusters. We execute the simulation along the sampling points and calculate the number of total execution cycles. Results from a 16-core simulation show that our proposed simulation technique gives us a maximum of 443x speedup with a 0.52% error and 218x speedup with 1.50% error on an average.**

***Keywords*-Multi/Many-core; Simulator; Compiler**

I. Introduction

The interaction among compilers, operating systems, and architectures is gaining in importance because it can facilitate power-efficient computing through a collaboration between hardware and software. Architecture simulators should play an important role in exploiting the new frontiers of such collaborations at early stages of the system development. However, much research has been hindered by time-consuming architecture simulators.

A considerable amount of research has been conducted to reduce the simulation time of architecture simulators. Some of them have proposed extracting simulation points, which need to be evaluated precisely. This approach combines two simulation modes to reduce simulation time. The one is the detail mode, which models the complete architectural state of a computer, such as its pipeline, memory hierarchy, interconnection network, and so on. The other is the function mode, which operates only instruction execution. The speed gap between the two simulation modes is more than two orders of magnitude. By simulating a small part of the program in the detail mode and the remainder in the fast functional mode, we can obtain the number of total execution cycles of an application program on a target multi/many-core processor in short simulation time.

SimPoint attempted to exploit the part of the program to be simulated in the detail mode by detecting execution phases in a program[1]. SMARTS framework applied a statistical sampling technique to the target program[2]. Both techniques are promising attempts to drastically reduce simulation time. However, both were originally developed to simulate a single-core architecture.

Several recent studies have tried to extend the sampling technique proposed by SMARTS to apply multi/many-core simulations[3], [4], [5]. One of the main challenges of this is overcoming thread skew at the function mode.

Carlson et al. and Ardestani et al. have proposed IPC calculation for each core in the function mode[3], [4]. On the other hand, Bryan et al. use the intervals between barrier synchronizations as the parallel execution unit[6]. Moreover, Carlson et al. use the barrier interval as a sampling unit to avoid thread skew[5]. Using barrier synchronization is a natural way to avoid thread skew for parallelized applications, since all threads are aligned by force at barrier synchronization points. This technique requires trace information to prepare sampling points using basically the same number of cores as the target machine for precise simulation.

In this paper, we use an iteration of a large loop, which includes the parallelizable part of the program, as a sampling unit. This is a coarser sampling unit than a barrier interval. Such an iteration includes all parallel primitives, such as thread fork and join, synchronizations, etc in addition to the ordinary computations in a program. Of course, all threads are aligned at the beginning of the iteration as a barrier synchronization is processed until the end of the iteration. Further, all important interactions among cores and shared hardware resources within such an iteration can be more naturally captured by the detail simulation mode.

A factor to take into account in using an iteration as a sampling unit is the choice of iterations to be simulated in the detail mode because large variation across iterations is anticipated. In this paper, we assume that for a typical parallelized application, the trend in cost transition over iterations is expected to be similar among different architectures and varying number of cores. Based on this assumption, all iterations of a large loop in a target parallelized program are clustered by X-means[7] by using the profile result from a real single-core machine. The sampling points from those clusters are then exploited. After sampling the simulation

978-1-5090-3531-1/16 $31.00 © 2016 IEEE
DOI 10.1109/MCSoC.2016.16

along the exploited sampling points, we calculate the total number of execution cycles in a multi/many-core environment.

The main contributions of this paper are as follows:

- We propose that an iteration of a large loop in a parallelized program be used as a sampling unit to avoid thread skew.
- We propose using the profiling result of a real single-core machine. This enables us to reduce the preparation time for deciding sampling points before a multi/many-core simulation.
- We show that the X-means method automatically decomposes iterations into appropriate clusters to minimize the number of sampling points by reducing the variance among clusters.
- We propose a simulation flow integrated with a parallelizing compiler.

The remainder of this paper is organized as follows. We describe in Section II the basic idea of behind our use of an iteration of large loops in a parallelized application as our sampling unit. Section III describes the proposed simulation framework. We present the experimental results in Section IV.

II. Motivation for Using a Large Loop in a Parallelized Application

In this section, we describe our motivation for using an iteration of a large loop in a parallelized application as a sampling unit by using an MPEG2 encoder as an example.

A. Large Loop in a Parallelized Program

Fig. 1 shows the main loop of an MPEG2 encoder program. The function "putseq" is the most time-consuming function. It has a loop that processes all frames in the relevant movie file. An iteration of this loop processes macroblocks, which are blocks of 16x16 pixels each that forms the basic component of a frame in MPEG2. The main loop in "putseq" can be parallelized at the macroblock level. This main loop is called a "large loop," and an iteration of this large loop will be used as the sampling unit of the simulation. All primitives for parallel processing, such as thread fork, thread-join, barrier synchronizations, and point-to-point synchronizations, are included in the loop body. This means the last barrier synchronizations in an iteration placed before the end of the iteration resolves thread skew prior to the next iteration.

B. Similarity in Program Behavior among Different Architectures

Fig. 2 shows the transition of execution cycles of an iteration in the large loop in an MPEG2 encoder on different architectures. Fig. 2(a) shows the result for an Intel Xeon Processor, Fig. 2(b) shows that for an Intel Itanium2, and Fig. 2(c) for an IBM Power5 processor, respectively. The

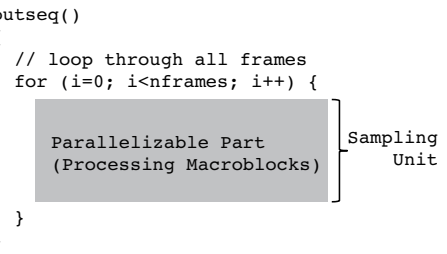

Figure 1. The large loop of an MPEG2 encoder program. The function "putseq" has a large loop that processes all frames. The body of this loop can be processed in parallel at the macorblock level.

detailed specifications of these processors are presented in Section IV. In each graph, the x- and y-axis represent the iteration and the clock cycles for each iteration, respectively.

Each graph has three groups of plots corresponding to operations for intra-coded (I)-pictures, predicted (P)-pictures and bidirectional-predicted (B)-pictures, respectively, as shown in Fig. 2(a). Furthermore, the graphs for the P-pictures and the B-pictures show a large variation due to the algorithms used in this program. The plots in Fig. 2(a), (b), and (c), have very similar shapes while the absolute values of the execution cycles of each are, of course, different.

In summary, two main observations of this section are:

- Thread skew can be avoided by using an iteration of a large loop in a parallelized program as a sampling unit.
- The trend in the transition of execution clock cycles over the iterations seems to be available as an architecture-independent parameter, since this appropriately represents program behavior.

We assess the second item in Section IV-A

III. Proposed Simulation Flow

In this section, we present our proposed simulation flow, which consists of two phases: the profiling phase, and the simulation phase. In the profiling phase, the number of execution cycles is profiled for each iteration of a large loop of the target program on a real single-core machine. The profile result is then grouped into clusters by X-means and the sampling points are extracted from the clusters. In the simulation phase, the architecture simulation is executed along the sampling points and the execution time is calculated. We also show a simulation framework integrated with a parallelizing compiler.

A. Profiling Phase: Profiling on a Real Machine and Exploiting Sampling Points

The profiling phase decides the sample size. Calculating sample size from the profile result on a real machine is based on the assumption discussed in Section II-B.

The first step of the profiling phase is profiling the iterations of target large loops on a real single-core machine.

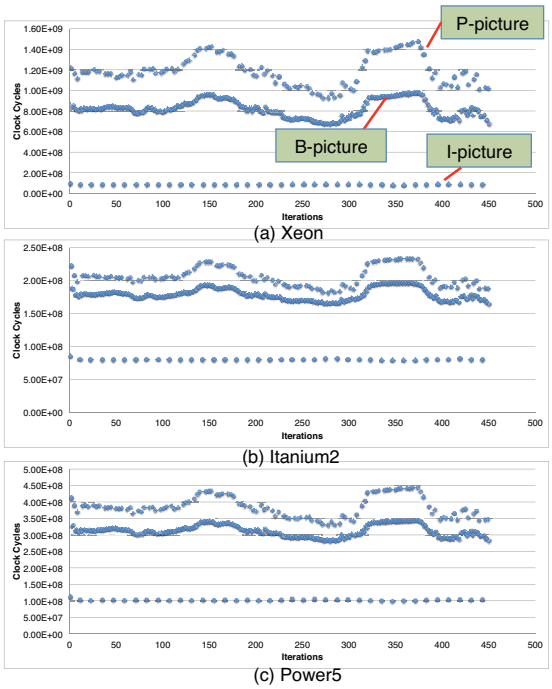

Figure 2. Trends in transition of execution cycles over iterations in an MPEG2 encoder on different architectures: (a) Xeon, (b) Itanium2, and (c) Power5. In each graph, the x-axis represents the iteration and the y-axis shows the clock cycles for each iteration.

A user specifies the target large loops, which include the parallelizable part of the target program.

The profiled iterations are then grouped into clusters to reduce the sample size by reducing the variation in execution cycles within each cluster. Another important role of clustering is specifying sampling points more appropriately than random sampling by exploiting sampling points from each cluster of iterations.

The sample size then is calculated from a cluster. The sample size n_i of the cluster Ci is calculated by the following expression, as shown in the literature [2]:

$$n_i = \left(\frac{z \cdot V_{xi}}{\varepsilon} \right)^2 \quad (1)$$

where z is the $100[1-(\alpha/2)]$-th percentile of the standard normal distribution ($(1-\alpha)$ is the confidence level), $\pm\varepsilon \cdot \bar{X}_i$ is the confidence interval of Ci ($\bar{X}_i$ is the mean of Ci), and V_{xi} is the coefficient of variation of Ci. Note that n_i becomes a known variable, since V_{xi} is calculated using the profile result. For the proposed technique, the sample size tends to be at most three after clustering, as detailed in Section IV-A, and statistical theory assumes the sample size is sufficiently large. However, such a small sample size is available as a good metric for clustering iterations and exploiting the sampling points in our technique. In this paper, we use $\pm 5\%$ as the confidence interval and 95% as the confidence level.

We use X-means[7], which is an extension of K-means, for clustering. X-means applies K-means ($k = 2$ in this paper) recursively until a predefined condition is satisfied. In this paper, X-means is applied based on the execution cycles of each iteration. For each iteration of recursive clustering, the sample size of each cluster is calculated according to (1). The recursive calculation of X-means is completed if the sample size does not decrease.

Finally, the sampling points are chosen from the clusters. In this paper, the sampling points are chosen from the beginning of the cluster.

B. Simulation Phase: Sampling-based Simulation and Calculation of Total Number of Execution Cycles

In the simulation phase, the sampling simulation is processed along the exploited sampling points. Codes for simulation mode-switching are embedded in the target parallelized program.

Then, the number of total execution cycles S is calculated by the following equation:

$$S = \sum_i \left(\frac{N_i}{n_i} \sum_j S_{ij} \right) \quad (2)$$

where N_i is the number of iterations in the cluster Ci, n_i is the sample size of Ci, and S_{ij} is the number of execution cycles of the jth sampling point in Ci. Other performance metrics, such as the number of cache misses, can be calculated by the same equation.

C. Simulation Framework Integrated with Parallelizing Compiler

Our proposed simulation flow can be integrated with parallelizing compilers, as shown in Fig. 3. In this simulation framework, at the Path-1 corresponding to the profiling phase, the source code of the target program is compiled and embedded profile code by the compiler for a real machine.

At the Path-2 corresponding to the simulation phase, the exploited sampling points are provided along with the source program to the compiler. The compiler generates parallelized code for the simulator.

This flow is implemented in the OSCAR parallelizing compiler[8], [9]. Note that this integration can also be applied to other kinds of parallelizing compilers, such as OpenMP compilers.

IV. Experimental Evaluation

We evaluate our proposed simulation technique in this section. In order to evaluate the applicability of profiling results for different architectures, the number of execution cycles of single-core execution on a machine is calculated by using the sampling points exploited from the profiling results of other machines. The error in the number of execution cycles of the multicore simulation is then calculated, follwoing which the speedup using our proposed technique is shown.

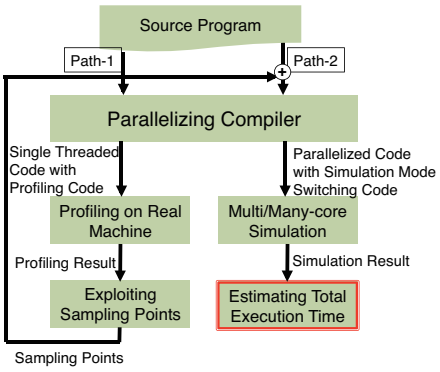

Figure 3. Simulation flow integrated with parallelizing compiler. In Path-1, the compiler generates single-threaded code and embeds profiling code. The profiling is carried out on a real machine. Next, in Path-2, the compiler generates parallelized code for a simulator and embeds simulation mode-switching code using sampling point information from Path-1.

A. Evaluated Applications

The following four applications are used in our experiments. All programs can be automatically parallelized by the OSCAR compiler. We choose these applications to confirm the collaboration between the simulation technique and the compiler.

183.equake (SPEC CPU 2000)
: The function "smvp()" is parallelized. An iteration of the loop, which calls smvp() inside it, is used as a sampling unit. The number of iterations of this loop is 3,855. The average and variance of the execution cycles of the first 250 iterations are larger than those of subsequent iterations.

179.art (SPEC CPU 2000)
: The function "match()" is parallelized. An iteration of the loop, which calls match() inside it, is used as a sampling unit. This iteration has convergence loops whose execution cycles affect the total execution cycles of the iteration. The number of iterations of the large loop is 500.

MPEG2 encoder (mediabench[10])
: We use 450 frames of a SIF size (352x240) movie as input data. An iteration of the loop in putseq() function is used as a sampling unit. These iterations have large variation as shown in Fig. 2.

470.lbm (SPEC CPU 2006)
: An iteration of the main loop in the "main()" function is used as a sampling unit. The number of iterations of this loop is 3,000. These iterations have small variability. However, every 64th iteration has more clock cycles than others. These iterations must be sampled to calculate the precise number of execution cycles.

Table I
PROCESSORS FOR APPLICABILITY OF PROFILE RESULTS IN DIFFERENT ARCHITECTURES

Processor	Intel Xeon X5670 (2.93GHz)	Intel Itanium2 9150M (1.6GHz)	IBM Power5 (1.5GHz)
L1 I-Cache	32KB, 4way	16KB, 4way	32KB, 4way
L1 D-Cache	32KB, 8way	16KB, 4way	64KB, 2way
L2 Cache	256KB, 8way	256KB, 8way	1.9MB, 10way
L3 Cache	12MB, 16way	24MB, 12way	36MB, 12way
OS	Linux 3.2.0	Linux2.6.16	Linux 2.6.9
Compiler	gcc 4.3.3	gcc 4.1.0	gcc 3.4.6

B. Applicability of Sampling Points in Real Machines

We evaluate here the clustering results and the sampling points derived from the profiling result on real machines. For profiling, Intel Xeon, Intel Itanium2, and IBM Power5 are used as shown in Table I. The instruction set architectures and micro-architectures of these processors are completely different. For example, Intel Itanium2 is EPIC, a kind of VLIW, architecture, whereas Intel Xeon and IBM Power5 are speculative out-of-order processors.

Tables II–V show the clustering results for 183.equake, 179.art, the MPEG2 encoder, and 470.lbm exploited from the profile results on the three architectures, respectively. For example, Table II shows that there are four clusters, C1 to C4, for 183.equake following clustering from the profile result of Intel Xeon. The total number of iterations of C1 is 78 and its sample size is 1, whereas the sample size is 4 for 3,855 iterations without clustering by X-means.

In Table II, there are four clusters each for Intel Xeon and Intel Itanium2, and seven clusters for IBM Power5. C1 and C2 correspond to the first 250 iterations, and C3 and C4 correspond to subsequent iterations for Intel Xeon and Intel Itanium2. Similarly, C1 to C5 of IBM Power5 correspond to the first 250 iterations. For all architectures, the first 250 iterations and subsequent iterations are clearly separated into different clusters by X-means.

In Tables IV–V, all cases have almost the same number of clusters among the three architectures. In particular, for 179.art shown in Table III, there are four clusters for each of the three architectures. The number of iterations for each cluster shows that all iterations are clustered in the same way for the different three architectures.

For 470.lbm, shown in Table V, all iterations with longer clock cycles than other iterations are captured by C3 and C4 (46 iterations in total) for all architectures.

Regarding the sample size, X-means reduces the sample size for 179.art and the MPEG2 encoder. For example, in case of Intel Xeon, 179.art has a sample size of 4 with X-means and 22 without it, and the MPEG2 encoder has a sample size of 13 with X-means clusering and that of 188 without it. This shows that X-means successfully forms clusters with a smaller variance than that of all iterations taken together.

The clustering results are then applied to the profiling

Table II
CLUSTERING RESULT OF 183.EQUAKE

Processor	Cluster	# of Iterations	Sample Size
Intel Xeon	w/o X-means	3,855	4
	C1	78	1
	C2	172	1
	C3	2,401	1
	C4	1,204	1
Intel Itanium2	w/o X-means	3,855	4
	C1	162	1
	C2	88	1
	C3	2,483	1
	C4	1,122	1
IBM Power5	w/o X-means	3,855	280
	C1	9	1
	C2	33	1
	C3	77	2
	C4	129	3
	C5	2	1
	C6	2,422	1
	C7	1,183	1

Table III
CLUSTERING RESULT OF 179.ART

Processor	Cluster	# of Iterations	Sample Size
Intel Xeon	w/o X-means	500	22
	C1	183	1
	C2	114	1
	C3	89	1
	C4	114	1
Intel Itanium2	w/o X-means	500	23
	C1	183	1
	C2	114	1
	C3	89	1
	C4	114	1
IBM Power5	w/o X-means	500	19
	C1	183	1
	C2	114	1
	C3	90	1
	C4	113	1

results to assess the applicability of the sampling points. To calculate the number of execution cycles, sample iterations are picked from the profiling result of the target architecture, but the samples are selected along sampling points exploited from another architecture.

Fig. 4 shows the errors in the number of execution cycles calculated using sampling points derived from different architectures. In this graph, for example, the three left-most bars show the error in calculation of the execution cycles of 183.eqauke on Intel Xeon, Intel Itanium2, and IBM Power5, the sampling points for which are exploited from the profile result of Intel Xeon.

For 183.equake, 179.art and the MPEG2 encoder, all calculation results show only a small error. For 183.equake in particular, the error is close to 0% except for cases involving IBM Power5 with sampling points from Intel Xeon and Itanium2. However, even for these cases, the error is only 2.72% and 3.50%, respectively. The highest error among these three applications is only 3.62% for the MPEG2 encoder on Intel Xeon with the sampling points from IBM Power5. These results show that sampling points can be effectively applied to different architectures.

Table IV
CLUSTERING RESULT OF MPEG2 ENCODER

Processor	Cluster	# of Iterations	Sample Size
Intel Xeon	w/o X-means	450	188
	C1	38	2
	C2	49	2
	C3	146	2
	C4	25	2
	C5	31	2
	C6	82	1
	C7	79	2
Intel Itanium2	w/o X-means	450	55
	C1	1	1
	C2	31	1
	C3	136	1
	C4	112	2
	C5	133	1
	C6	37	1
IBM Power5	w/o X-means	450	86
	C1	22	1
	C2	29	1
	C3	109	1
	C4	59	2
	C5	132	1
	C6	16	1
	C7	83	1

Table V
CLUSTERING RESULT OF 470.LBM

Processor	Cluster	# of Iterations	Sample Size
Intel Xeon	w/o X-means	3,000	3
	C1	2,925	1
	C2	29	1
	C3	24	1
	C4	22	1
Intel Itanium2	w/o X-means	3,000	5
	C1	1,455	1
	C2	1,499	1
	C3	2	1
	C4	44	1
IBM Power5	w/o X-means	3,000	6
	C1	50	1
	C2	2,904	1
	C3	41	1
	C4	5	1

For 470.lbm, errors using Intel Xeon are sufficiently low, such as 2.05% using its own sampling points, 1.06% using Itanium2 and 0.57% in IBM Power5. However, errors with Intel Itanium2 and IBM Power5 with sampling points from Intel Xeon are higher than 5.00%, (6.57% and 5.62%, respectively). Similarly, the error with IBM Power5 is 6.68% when sampling points from Intel Itanium2 are used. These relatively high errors are due to insufficient sampling points for Itanium2 and Power5. For these architectures, the first iteration of the loop is selected as the sampling point in the cluster C1 for Intel Xeon and Intel Itanium2 as shown in Table V. This iteration has a smaller number of execution cycles than other iterations involved in the same cluster. This can be avoided by changing the parameters z and ε in (1),

Table VI
TARGET ARCHITECTURE FOR SIMULATION

Instruction Set	SPARC V9
Pipeline	In-order, single-issue, 8-stage pipeline
# of Cores	1, 4, 8, 16
L1 I-Cache	32KB, 2way
L1 D-Cache	32KB, 2way
L2 Cache	512KB/core, 2way
Latency	L1: 1 clock cycle, L2: 5 clock cycle, Main Memory: 68 clock cycle
Cache Coherence Protocol	MOESI

or by selecting sampling points other than from the first iteration of the cluster.

C. Error of Calculation

We assess here the error in the multicore simulation. The specification of the simulated multicore architecture is shown in Table VI. The processors used here had 1, 4, 8 and 16 cores. The specification of the architecture is assumed to be used in embedded areas like mobile devices. We develop a cycle-accurate architecture simulator. A user program on this simulator can change the two simulation modes, the detail mode and the function mode, through a system call.

The sampling points used in this evaluation are exploited from the profiling results on Intel Xeon. In order to attain the number of total execution cycles in detail mode within a feasible simulation time, the target loops in each program are decomposed into multiple chunks, each of which consists of 5–50 iterations depending on the simulation time. One iteration is added to each chunk as a warm-up iteration. The chunks are then simulated in parallel, as in the literature[6].

In order to show the robustness of the proposed technique, the first 270 iterations were simulated for 183.equake, since the variation in iterations is very small after the 250th iteration, while the number of those iterations is very large (such as 3,605). The error in 183.equake largely depends on these 3,605 stable iterations, if all iterations are simulated.

Fig.5–8 show the errors in the calculated number of execution cycles, L1 cache misses and L2 cache misses for 183.equake, 179.art, the MPEG2 encoder and 470.lbm, respectively. The effect of warm-up is also evaluated. In these figures, 0-warmup, 1-warmup and 2-warmup stand for no-warm-up, one warm-up iteration and two warm-up iterations added prior to a sample iteration. The simulation model of the warm-up mode is the same as that of the detail mode in this paper.

The graphs show that the errors in the calculation of execution cycles for all programs are under 5.00% even without warm-up iterations, for all number of cores used. Regarding cache misses, when at least one warm-up iteration is added, the errors for almost all programs are also under 5.00% (The cases of programs exceeding 5.00% will be discussed later). The detailed discussion of each program is as follows.

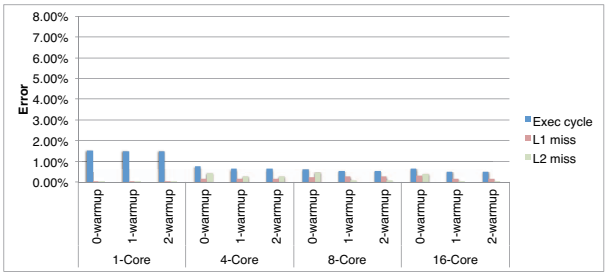

Figure 5. Error in Calculated Execution Cycles and Cache Misses of 183.equake.

Fig. 5 and Fig. 6 show that the proposed technique achieves very low error for 183.equake and 179.art, even without warm-up. For the case of 183.equake where 16 cores are used without warm-up, the error in the calculation of the number of total execution cycles, L1 cache misses and L2 cache misses are 0.67%, 0.31% and 0.41%, respectively. The four sampling points result in a sufficiently low error for this case. Similarly, for 179.art where 16 cores are used without warm-up, the number of total execution cycles is 0.002% with also only four sampling points.

Fig. 7 shows that the proposed technique leads to a sufficient low error in the calculation of the number of execution cycles for the MPEG2 encoder. The highest error is 2.83% for eight cores with 2-warmup, and only 0.59% in the case of 16 cores with 1-warmup. However, regarding the number of L2 cache misses, 0-warmup yields high errors, such as 53.39% for eight cores and 64.12% for 16 cores. The main reason for these high errors is the relatively small number of L2 cache misses. For 16 cores, the number of L2 cache misses per iteration is 17.3×10^3 while the number of execution cycles is 95.7×10^6. By adding a warm-up iteration, the errors can be reduced to 1.38% for eight cores and 6.87% for 16 cores.

Fig. 8 shows that error in calculation of execution cycles for 470.lbm increases along with the increasing number of cores, while error in the calculation of the number of cache misses remains low. For example, for the case with no warm-up iteration, the error was 2.75% for eight cores and 4.94% for 16 cores. This is due to the differences in the execution cycles between the first iteration and other iterations, as described in Section IV-B. The clustering result of Intel Xeon, shown in Table V, shows that the cluster C1 garners 97.5% of all iterations (2,925 of 3,000 iterations). The first iteration is selected as the sampling point of C1 in this evaluation. However, the number of execution cycles of C1 decreases with an increasing number of cores. This causes the increased error in the calculation of execution cycles in the case of eight cores and 16 cores.

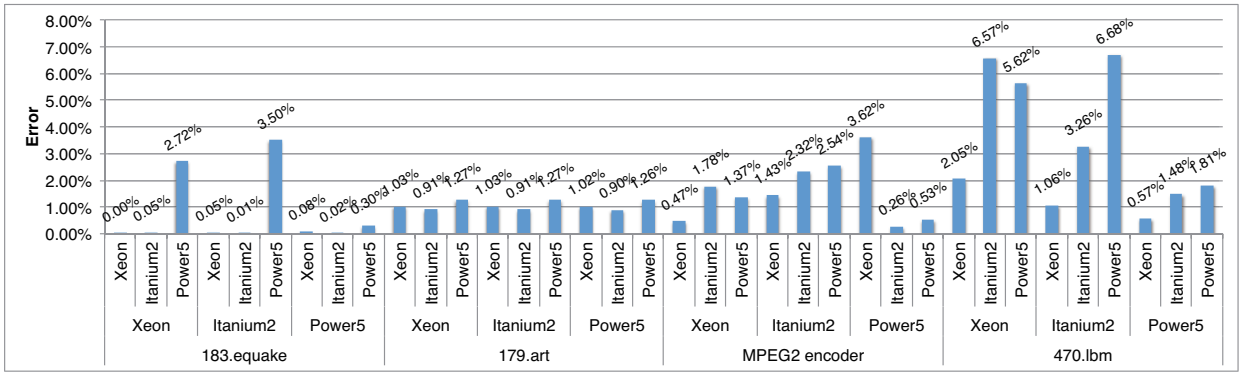

Figure 4. Error in calculated execution cycles. Each bar shows an error in execution cycles when sampling points exploited from another processor are applied to the target processor for each application. For example, the left-most bar shows an error in estimated execution cycles calculated by sampling points exploited from the profile result of Intel Xeon. Similarly, the next bar shows the error with Intel Itanium2 with sampling points from Intel Xeon.

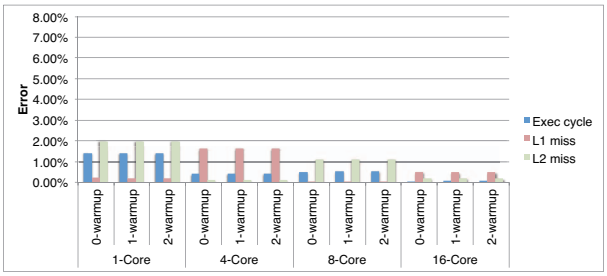

Figure 6. Error in Calculated Execution Cycles and Cache Misses of 179.art.

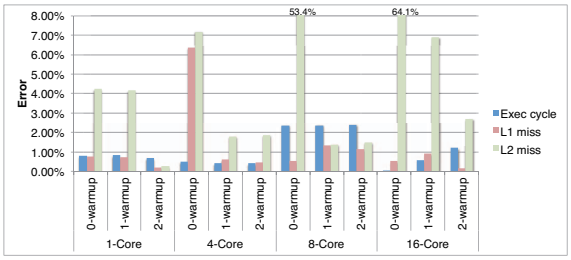

Figure 7. Error in Calculated Execution Cycles and Cache Misses of MPEG2 encoder.

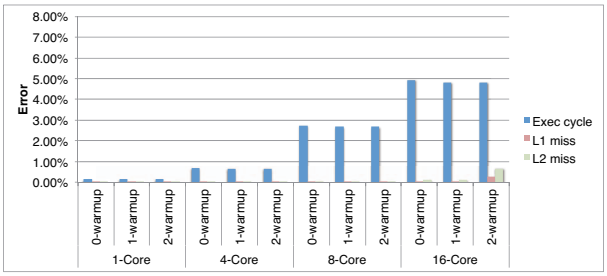

Figure 8. Error in Calculated Execution Cycles and Cache Misses of 470.lbm.

D. Speedup Using The Proposed Technique

The speedup using the proposed technique is assessed under the assumption that all iterations are sequentially simulated in the detail mode. In order to calculate the speedup, we first calculate the speed ratio of the function simulation mode to the detail simulation mode by measuring the simulation time of two iterations in the target program on Intel Xeon 5650 (2.67GHz). Speedup is then calculated by using the speed ratio and the sample size of the program. When warm-up iterations are required, they are included in the detail mode.

Table VII–X show the speedup of each program. In these tables, "F / D" stands for the speed ratio of the function mode against that of the detail mode. "Speedup (0W)," "Speedup (1W)" and "Speedup(2W)" represent the speedup in case of no warm-up iterations, one warm-up iteration, and two warm-up iterations, respectively. The speed ratio tends to depend on the L1 miss rate and the L2 miss rate, as well as the number of cores, since these values affect IPC, which is not modeled in the function mode.

The tables show that the proposed technique provides a speedup of two or three orders of magnitude. In this section, taking into account the errors in the number of cache misses, as discussed in Section IV-C, we focus on the speedup in the case of 16 cores with one warm-up iteration.

Table VII and X show significant speedups, such as 443x speedup for 183.equake and 327x speedup for 470.lbm. These high values are obtained because the relevant programs have a large number of iterations with a small sample size (only four for both cases).

Table VIII shows, for 179.art, 81x speedup, which is lower than for 183.equake and 470.lbm. This is because the number of iterations of the target large loop is 500 and the sample size is four. Similarly, Table IX shows 19x speedup for the MPEG2 encoder. The reason of this low speedup is due to low speed ratio caused by a low cache miss rate, such as 1.14% for L1 and 10.16% for L2, respectively, in addition to a sample size of 13 for 450 iterations.

In summary, 218x speedup is achieved on an average using our proposed method (57x speedup on a harmonic

Table VII
SPEEDUP OF 183.EQUAKE.

# of cores	L1 miss (%)	L2 miss (%)	F / D	Speedup (0W)	Speedup (1W)	Speedup (2W)
1	17.26	17.98	941	476	382	319
4	16.17	17.84	666	394	327	280
8	15.33	17.66	818	443	360	303
16	13.73	17.52	1428	576	443	360

Table VIII
SPEEDUP OF 179.ART

# of cores	L1 miss (%)	L2 miss (%)	F / D	Speedup (0W)	Speedup (1W)	Speedup (2W)
1	57.58	59.14	743	107	75	58
4	13.20	50.81	979	111	77	59
8	5.83	54.41	1587	116	79	60
16	1.87	64.69	2844	120	81	61

Table IX
SPEEDUP OF MPEG2 ENCODER

# of cores	L1 miss (%)	L2 miss (%)	F / D	Speedup (0W)	Speedup (1W)	Speedup (2W)
1	2.58	5.02	106	26	19	15
4	1.91	9.58	95	26	18	14
8	1.60	10.70	104	26	19	14
16	1.14	10.16	129	27	19	15

Table X
SPEEDUP OF 470.LBM

# of cores	L1 miss (%)	L2 miss (%)	F / D	Speedup (0W)	Speedup (1W)	Speedup (2W)
1	19.32	20.41	282	205	170	146
4	15.64	20.41	338	233	189	159
8	15.16	20.42	617	339	253	202
16	14.04	20.43	1385	487	327	327

mean).

V. CONCLUSION

We proposed a sampling-based technique using profiling data on a real machine for architecture simulators targeting parallelized applications to reduce the simulation time.

The proposed technique consists of two phases: the profiling phase and the simulation phase. In the profiling phase, large loops that contains a parallelizing part are specified in the target program. The execution cycles for each iteration in these large loops are then profiled on a single-core machine. The profile results are decomposed into multiple clusters and sampling points are exploited from each cluster by X-means. Following this, the simulation phase is carried out by using the exploited sampling points.

Experiments showed that one architecture can exploit sampling points from other architectures. The results showed that the number of execution cycles can be calculated within a 5% error for almost all cases by using sampling points from the other architectures. The speedup and error in the architecture simulation were then evaluated. The evaluation results showed that the proposed technique provides a maximum of 443x speedup with a 0.52% error, and 218x speedup with a 1.50% error on an average.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number JP15K00085.

REFERENCES

[1] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, 2002, pp. 45–57.

[2] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03)*, 2003, pp. 84–97.

[3] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sampled simulation of multi-threaded applications," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2013)*, 2013, pp. 2–12.

[4] E. K. Ardestani and J. Renau, "Esesc: A fast multicore simulator using time-based sampling," in *Proceedings of IEEE International Symposium on High Performance Computer Architecture (HPCA 2013)*, 2013, pp. 448–459.

[5] T. E. Carlson, W. Heirman, , K. V. Craeynest, and L. Eeckhout, "Barrierpoint: Sampled simulation of multi-threaded applications," in *Proc. IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS2012)*, 2012, pp. 117–126.

[6] P. D. Bryan, J. A. Poovey, J. G. Beu, and T. M. Conte, "Accelerating multi-threaded application simulation through barrier-interval time-parallelism," in *Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2014)*, 2014.

[7] D. Pelleg and A. Moore, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco: Morgan Kaufmann, 2000, pp. 727–734.

[8] M. Obata, J. Shirako, H. Kaminaga, K. Ishizaka, and H. Kasahara, "Hierarchical parallelism control for multigrain parallel processing," *Lecture Notes in Computer Science*, vol. 2481, pp. 31–44, 2005.

[9] K. Kimura, M. Masayoshi, H. Mikami, T. Miyamoto, J. Shirako, and H. Kasahara, "Oscar api for real-time low-power multicores and its performance on multicores and smp servers," *Lecture Notes in Computer Science*, vol. 5898, pp. 188–202, 2010.

[10] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "Mediabench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. of 30th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO30)*, December 1997, pp. 330–335.