Multi-granularity Sampling for Simulating Concurrent Heterogeneous Applications

Melhem Tawk
University of Valenciennes and Hipeac Network of Excellence
59313 VALENCIENNES Cedex 9, France
melhem.tawk@univ-valenciennes.fr

Khaled Z. Ibrahim
IRISA/INRIA, Campus de Beaulieu
35042 RENNES, France
kibrahim@irisa.fr

Smail Niar
INRIA Lille-Nord-Europe and Hipeac Network of Excellence
40, Avenue Halley
59650 VILLENEUVE d'ASCQ
Smail.Niar@inria.fr

ABSTRACT

Detailed or cycle-accurate/bit-accurate (CABA) simulation is a critical phase in the design flow of embedded systems. However, with increasing system complexity, full detailed simulation is prohibitively slower than the hardware being simulated. In this paper, we present an approach that uses the sampling technique to speed up the design flow of Multiprocessor System-on-Chip (MPSoC) systems. Based on the dynamic behavior of the applications running concurrently, our method dynamically chooses between multiple granularities of the sampling phase. The similarities of the execution phases for all possible granularities are first analyzed, then transitions between phase overlaps are discretized. To facilitate the detection of repetitions, one phase, with an appropriate granularity, is chosen per process. Unlike most other proposals, the associated performance is usually accurate enough not to need repeated resampling. The use of checkpointing in conjunction with our approach is simplified because the amount of the needed disk space is significantly reduced. Experimental results show that the simulation of concurrent heterogeneous applications can be accelerated by a factor of up to 60x, while maintaining an average performance estimation error lower than 5%.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems; C.4 [**Performance of Systems**]: measurement techniques, modeling techniques; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms

Design, Performance, Measurement

CASES'08, October 19–24, 2008, Atlanta, Georgia, USA.

Keywords

Multiprocessor System-on-chip, simulation acceleration, simulation sampling

1. INTRODUCTION

Embedded and mobile system design relies heavily on simulators to evaluate and validate new platforms before implementation. This process of Design Space Exploration (DSE), consists in evaluating the performance and the power consumption of a large possible set of system configurations at the micro-architectural level. Nevertheless, as technological advances allow the realization of more complex circuits, simulation time is considerably increasing. Consequently, conventional cycle-accurate simulation tools risk provoking bottlenecks in the design flow of future high performance Multiprocessor Systems-on-Chip (MPSoC).

In this paper, we build upon the use of application sampling technique [17, 21] to reduce simulation time for parallel heterogeneous embedded applications. By "heterogeneity," we refer to large differences in the behavior and the performance of applications running concurrently rather than to the changes in behavior within each application phases. The basic principle of sampling is to simulate only a small representative set of the application intervals, called phase samples, in order to estimate the overall application performance [6, 11, 12, 13, 14]. Intervals with known performance estimates are either skipped, thus requiring checkpoints of the system state after the skipped intervals, or simulated with simple fast-forwarding (or functional simulation). Checkpointing is usually associated with large disk-space overhead, while fast-forwarding is usually associated with additional simulation cycles.

One of the complexities associated with the use of sampling to accelerate simulation in multiprocessor systems is the determination of the representative parallel phases that are executed simultaneously by different processors. As parallel phase overlaps depend on the architectural configuration, it is impossible to determine these concurrent application phase overlaps a priori. Additionally, phase execution times depend on the preceding executed phases in cases of inter-phase dependency. Inter-phase sensitivity is architecture-dependent and cannot be taken into account during the static profiling step of the application. For large samples, this relationship may have small influence on the simulation accuracy. However, small samples are gener-

ally preferred in order to obtain the desired high simulation speedup to explore a large design space. Such difficulties mostly arise in MPSoC systems due to the high level of heterogeneity in the concurrently running applications.

Among recently proposed approaches, the co-phase matrix approach [6] relies on sampling overlaps and consequently every overlap measurement is taken multiple times to achieve an acceptable level of accuracy. This approach is extended to support un-simulated overlaps by Biesbrouck et al. [5]. The main difficulty with this approach is the determination of the percentage of samples needed to achieve an acceptable level of accuracy. This is especially true for applications with extremely large behavioral differences.

Another recent approach, *adaptive sampling* [18] tries to establish deterministic overlap scenarios by adopting two complementary mechanisms: 1) discretization of the overlap boundaries using simulation barriers that define when to start and to end overlaps, and 2) detection of repetition in the simulation by matching multi-phase strings from each process under consideration. The main objective of discretizing overlaps of phase strings is to guarantee sampling accuracy by generating precise overlaps, allowing accurate estimation of performance for each overlap. One advantage of this approach is that it does not require the re-sampling of recurrent overlaps of phase strings because of its precise performance estimates.

Simulating many heterogeneous applications simultaneously using the approaches mentioned above faces several challenges. For co-phase matrix approach, the sampling transition increases with the number of overlapping threads, as shown in Figure 1.a. The overlap periods are also shortened as the number of joint scenario increases for the three consecutive cophases, labeled a_1;b_1;c_1. The idea that each encountered phase overlap should be taken as representative of all such overlap scenarios is severely tested. For instance, the overlap a_1;b_1;c_1, in Figure 1.a, appears four times. All represent different joint phase scenarios and different number of simulated instructions; thus, it is not safe to assume that the execution of a phase is homogeneous or that any part of the phase is representative of the whole phase. Generally, SimPoint [17] detects similar phases, based on their basic block distribution, but cannot guarantee that the behavior within a phase is homogeneous. A special case of phase homogeneity may appear associated with large regular loops. In general, the more joint scenarios, the more the samples needed to achieve acceptable accuracy.

On the other hand through, *adaptive sampling* [18], shown in Figure 1.b, can easily achieve good levels of accuracy but the acceleration of simulation is reduced because *adaptive sampling* is very conservative when detecting similarity. The main source of this conservatism is the phase string matching process that is associated with detecting repeated overlaps. As the number of threads increases, the phase string length also increases and thus detecting repeated scenarios becomes less likely.

In the multigranularity sampling approach (Figure 1.c), proposed in this paper, we maintain the accuracy of the process defining discrete overlaps. In this way, we guarantee the detection of repeated scenarios and we maintain the simplicity by reducing the phase string length. Our approach involves the following steps: each thread is analyzed offline for multiple levels of granularity that can be encountered at runtime. Fine-granularity is usually used as a base and then coarser granularities are composed and analyzed. This one-time low-cost offline analysis allows simulation times to be decreased as follows: at the end of each discrete simulation point, the appropriate sample granularity is chosen for each process so that only one sample is contributed by each process while maintaining a perfect overlap of the samples from all processes. This strategy of one sample per-process allows repeated behaviors to be detected much more easily. The simulation time is greatly reduced because all repeated behaviors are detected. This approach insures a high level of accuracy because of the discrete overlap of appropriate-size samples from each application and because of the precise offline analysis of these multi-granularity samples. Our approach offers significant gains when simulating large number of processes, with the expected performance heterogeneity.

To leverage the benefit of our sampling technique, we investigate minimizing the overhead associated skipping unsimulated intervals based on checkpointing. We show that multigranularity sampling allows choosing a few representative checkpoints, thus reducing the space needed to hold these checkpoints. In fact, the representative checkpoints can easily reside in physical memory, due to the small memory space needed, during simulation instead of having them on hard-drive. Consequently, the process of restoring the system state based on the few memory-resident checkpoints becomes very efficient.

In this paper, we make two main contributions; we define multi-granularity approach for speeding up the simulation of concurrent heterogeneous parallel applications; second, we demonstrate that, by using our method, checkpointing can be simplified by reducing the overhead associated with storing system states.

The rest of this paper is organized as follows. An overview of related work on existing simulation acceleration techniques is presented in Section 2. Then, our multi-granularity approach sampling is detailed in Section 3. The simplification of checkpointing technique based on our approach is presented in Section 4. The evaluation of the proposed approach, in terms of estimation accuracy and simulation speedup, is given in Section 5. In that section, we also compare our approach to the adaptive sampling method. Finally, we present our conclusions in Section 6.

2. RELATED WORK

Design Space Exploration (DSE) is usually accelerated in two ways:

1. By reducing the number of architectural alternatives that need to be evaluated and steering the search towards the most promising configuration part of the design space using for instance smart heuristics [8, 1], or

2. By reducing the time needed for performance and power consumption evaluations associated with each design alternative.

In this paper, we deal with the second alternative. In this context, several approaches are possible. In Del Valle et al. [19] and J. Schnerr et al. [16] simulation are accelerated by using an FPGA-based emulation framework. Both of these two works allow a large range of statistics to be extracted rapidly from the MPSoC components. Moreover,

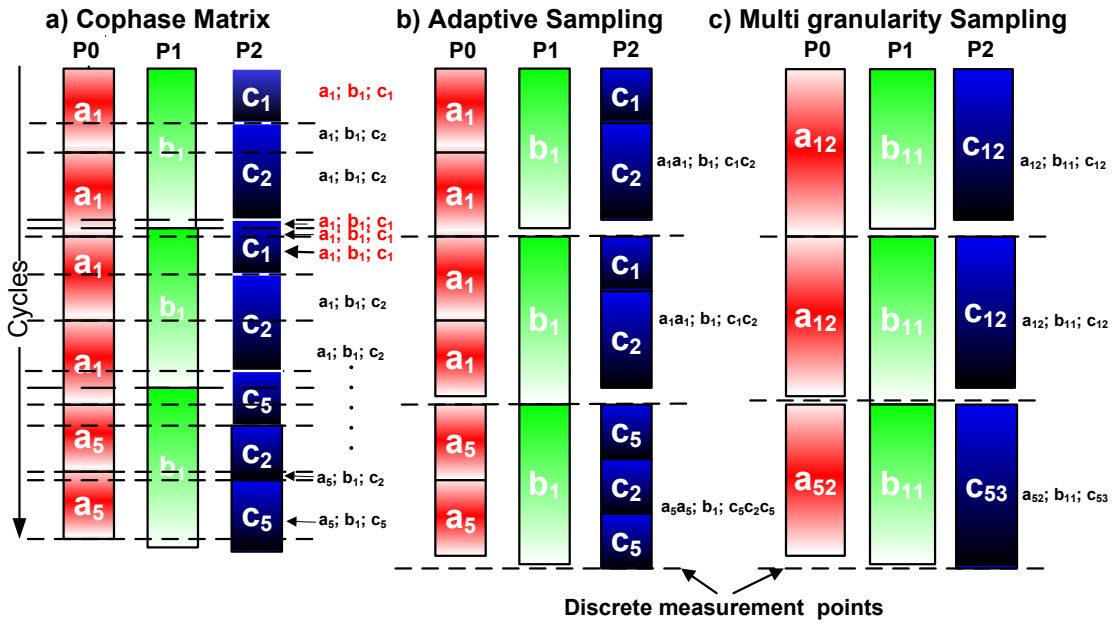

Figure 1: Different approaches to multiprocessor sampling. Each block corresponds to a phase of the application.

the results obtained are accurate and offer good acceleration factor, but require a long development phase and an expensive FPGA development environment. Statistical simulation [15], analytical modeling [10] and transactional level modeling (TLM) [7, 2] have also been widely studied recently. These approaches have some interesting features but are very complex to implement for MPSoC. In addition, they offer generally a reduced speedup factor.

Simulation by application sampling is another approach to reduce simulation time. In such methods, only a small portion of the application is simulated in detail. The rest of the application is either skipped or simulated at the functional level, thus reducing the total simulation time. Two sampling methods, SMARTS [21] and SimPoint [17], are commonly used to explore architectural designs. SMARTS relies on very short periodic samples of detailed simulation. SimPoint, on the other hand, uses longer representative samples of detailed simulation. Each sample corresponds to a distinct application phase. To determine these phases, the application's run is divided into intervals of either a fixed or variable number [13] of instructions that are classified according to the similarity of the basic block distribution. SimPoint accelerates the simulation by exploiting the fact that intervals classified as similar (same phase) have almost identical behavior, making it enough to simulate only one representative sample from each phase in CABA. Kihm et al. [12] combined the strengths of the above two methods in a technique called PGSS-SIM. By tracking phase behavior during simulation, intelligent decisions can be made on the intervals to perform short periods of detailed simulation. Recently, several studies have applied the sampling technique to multi-context architectures, such as multi-thread processors or multi-processor architectures.

The method proposed by Biesbroucky et al. [3], first combines phases into what is called a co-phase, and then simulates the recurring co-phases once. Because these phases are not generally homogeneous, overlapping the same couple of phases results in multiple performance estimates. Multiple samples of co-phase overlap are reportedly needed [6] to achieve accurate performance predictions. However, for a given simulation accuracy, it is difficult to determine the number of samples needed a *priori*. Namkung et al. [14] applied this co-phase approach to multi-processor architectures. They also noticed that simulation acceleration diminishes when there is a rapid increase in the number of phase combinations. In their solution, the number of simulation samples is reduced by synthesizing samples from similar phase combinations. Kihm et al. [11] extended their original PGSS-SIM method to multi-context simulation in a technique called CoGS-sim. The idea behind this technique is that, each program is fast-forwarded based on its IPC, measured during the last detailed sample.

In order to eliminate the need of resambling while assuring a high level of accuracy in multiprocessor performance estimation, Tawk et al. [18] proposed the adaptive sampling approach. Both multi-granularity sampling and adaptive sampling try to discretize the overlaps between overlapping applications running concurrently on the system. This discrete overlap allows accurate performance estimates to be obtained without resampling the same overlap. The adaptive sampling approach consists of the following two steps:

1. **Program tracing and phase identification:** This step is accomplished once for all the evaluated configurations in the DSE. The interval size must be specified by the user, for instance 50K instructions.

2. **Generation and use of clusters to accelerate the simulation:** For each processor, consecutive simulated phases are combined together to form a phase string. A cluster of strings (CS) consisting of p parallel strings is generated dynamically, where p is the number of processor cores. Each new CS executed at

the CABA level is allocated an entry in the cluster of string table (CST). The simulation begins by searching whether or not one entry in the CST corresponds to the next phases from the program tracing. If a match is found, then the next CS is skipped. If no match is found, then a detailed simulation is performed. At the end of every simulation interval, each processor estimates the remaining cycles so that all other processors finish their intervals currently in progress. If the number of cycles remaining is less than a given threshold for all processors, the processor stops the simulation and waits at a simulation barrier. These simulation barriers represent boundaries between clusters.

Clusters containing the same parallel phase strings have the same behavior and thus give the same performance. In the next sections, we will compare our proposal with this scheme.

3. MULTI-GRANULARITY SAMPLING FOR MPSOC

The multi-granularity sampling approach responds to the need to execute different number of instructions from concurrent heterogeneous applications during the same simulation cycles. Traditionally for each application, a single phase trace is generated and a phase classification technique (such as Simpoint) is used to detect similarity. In our approach, we extend the analysis to all the granularities that can be encountered during runtime. The same start point of program execution is associated with multiple classifications based on the granularity of the phase (instruction count), as will be detailed in the following subsections.

3.1 First step: Phase matrix creation

This first step is realized once during all the DSE process. It starts by generating a phase-ID traces for each application individually using a rapid functional simulation. These traces are neither dependent on the concurrent processes on the other processors, nor are they dependent on a specific MPSoC architectural configuration. This first stage is realized as follows: Each application is decomposed into intervals of granularity order of 1 (50K instructions in the experiments). For each interval, a basic block vector (BBV), containing the frequencies of executed basic blocks in the corresponding interval, is generated. A phase-ID trace is then generated by examining the similarity between these BBVs using SimPoint [17] classification. Other classification tools can also be used for this stage.

Using the same starting points (i.e, a discretization point of 50K instructions), overlapping samples of coarser granularity (100K, 150K, ..., etc) instructions are formed, and SimPoint is used repeatedly to detect similarity for each phase granularity. Specifically, for order g granularity, the basic block vectors (BBVs) of order 1 granularity are combined with the g-1 BBVs that succeed them, by adding the frequencies of each basic block of the corresponding g BBVs. Thus, new BBVs corresponding to intervals of g*50K instructions are generated and then processed to classify phases of execution.

As shown in Figure 2, the multi-granularity phase matrix for the application "a" in Figure 1 is generated. Each column corresponds to a granularity order. Thus, if there are G granularities, there will be G columns. Each line in the multi-granularity matrix corresponds to one starting point in the program. The matrix element belonging to line i and column j, corresponds to the phase whose size is j*50K instructions and whose starting point is the (i-1)*50K^{th} instructions. An example with four orders of granularity is given in Figure 2.a and Figure 2.b. Figure 2.a shows the formation of the intervals of each granularity order. Then, Simpoint is used to detect the similarity in the level of each granularity in order to generate the phase matrix. Each interval, in Figure 2.a, is represented in Figure 2.b by a phase which corresponds to the representative interval. For instance the interval a_{31} which has the granularity of order-1 and starts at the second discretization point is represented by the phase a_{11} in the matrix. The corresponding phase a_{11} belongs to the third line and to the first column in the matrix. The first stage is executed once for all the granularities expected during runtime and a rapid functional simulation is used to obtain phase matrices in a few minutes. In our experiments, we consider granularities up to 20. This requires less than 10 minutes of profiling per application.

Table 1: A multi-phase cluster table (MPCT) containing 4 MPCs and their metrics. This MPCT corresponds to Figure 2.b and Figure 2.c. "g_i" corresponds to the phase granularity (in K inst) for processor *i*.

MPC	g_0	g_1	g_2	K-Cycles	Energy(mJ)	Repetition
a_{12}, b_{11}, c_{12}	100	50	100	280	104	3
a_{52},b_{11},c_{53}	100	50	150	487	190	1
a_{21},b_{11},c_{12}	100	50	150	277	150	1

3.2 Second step: multi-phase cluster generation

The second stage uses the multi-granularity phase matrices that have been generated in the first step. Phases that are executed in parallel by processors are combined together to form a multi-phase cluster (MPC). These parallel phases are determined dynamically as follows: An MPC contains p parallel phases, where p is the number of processors in the MPSoC. Each new MPC is assigned an entry in the multi-phase cluster table (MPCT)(see Table 1). MPC containing the same parallel phases have the same behavior, and thus can be skipped during simulation after estimating their performance once.

In our approach, we adopted the discretization of the overlapping execution phases, as proposed by Tawk [18]. In order to generate the MPCs, the granularity of each of the p phases is determined dynamically. Whenever a process reaches the boundary of 50K instructions, a decision is made as to whether to continue the simulation or to stop to form an MPC. The number of cycles needed to finish the current interval for each processor is calculated.

If the maximum number of cycles is less than a certain predefined threshold TWSS[1], then the approach enters the cluster formation mode, in which each process tries to finish the current phase and then waits for the other processes. As the IPCs (Instructions Per Cycles) of each application executing in parallel are not the same, the number of instructions executed by each processor in an MPC is gener-

[1]Threshold Waiting at Simulation Synchronization is denoted by TWSS.

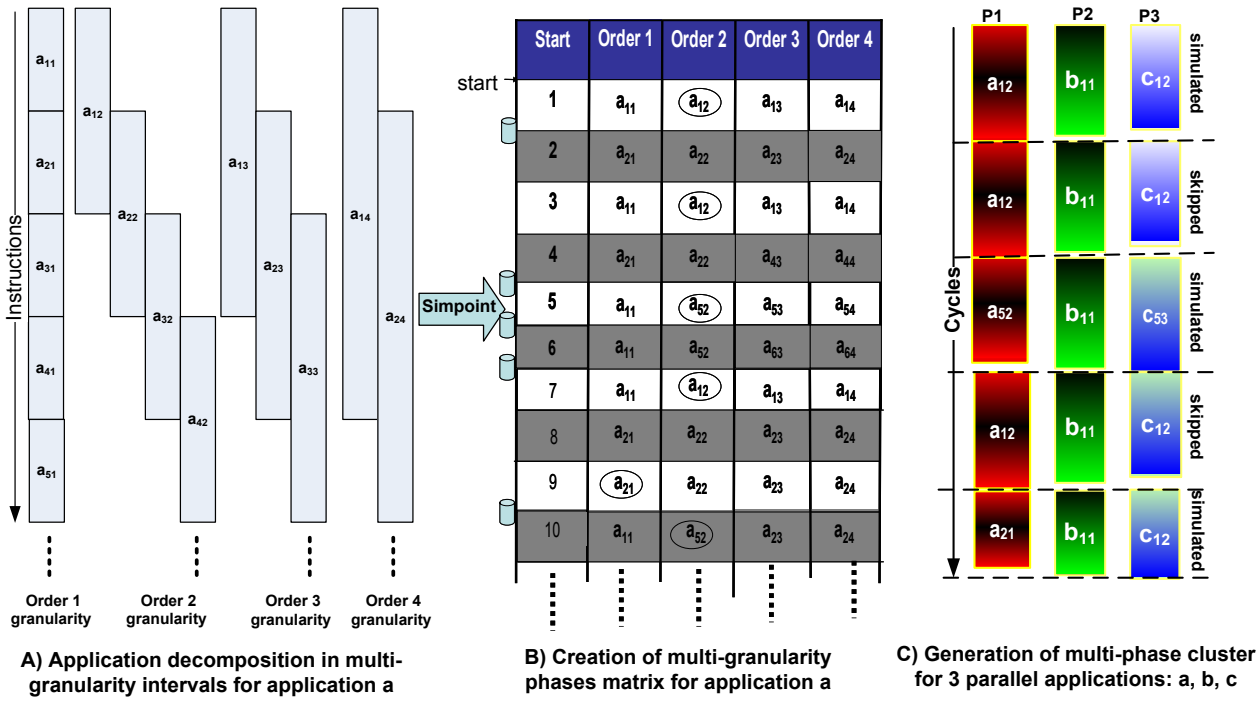

Figure 2: Multigranularity Phase matrix generation for the application *a* executed by processor P1 (Fig. 2.A and Fig. 2.B) and the simulation acceleration for three applications a, b, and c (Fig. 2.C).

ally different but the number of instructions executed is a discrete multiple of the base granularity size. Phase identification is dependent on the starting point of the simulation and the number of instructions simulated. The 5th MPC of Figure 2.c in the level of P1, starts at the 8th discetization point (after the execution of fours phases belonging to order-2) and the number of instructions executed is equal to the interval size of order-1. Thus the phase of P1 in that MPC is a_{21} belonging to line 9 and column 1 in the matrix. In the MPCT, the combined phase identifiers, which are executed in parallel, represent a unique MPC identifier and MPCs with the same identifiers are predicted to have the same performance. Once a new MPC is simulated, a new entry in the MPCT is created for it. The metrics for the new MPC (e.g., the number of instructions executed or the granularity order for each processor, the number of cycles, the energy consumed, memory traffic, internal bus traffic and cache miss rate ...) are then recorded in this table.

After each skipped or simulated MPC, all the entries in the MPCT are checked against the multi-granularity phase matrices to find a match with the next application phases in the phase matrices. For each simulated MPC in the MPCT, only one access is realized to each phase matrix. If a match is found for an MPC in the MPCT and the successive phases in the matrices, the search is stopped. Then each phase of the MPC is skipped by the corresponding processor and the corresponding repetition entry in the MPCT (see Table 1) is incremented. If no match is found, a detailed simulation is performed until a new MPC is formed. As two MPCs containing the same phases are assumed to have the same performance, the metrics of these MPCs are associated with the metrics of the first simulated MPC. Thus, the performance of the whole application can be estimated based on the metrics of the simulated MPCs and their relative contribution to the applications' execution. In Table 1, after simulating the first MPC (a_{12};b_{11};c_{12}), this MPC recurs. Since it already exists in the MPCT, the repetition entry is simply incremented, and the repeated MPCs are not simulated in details.

4. CHECKPOINTING FOR SYSTEM IMAGE CONSTRUCTION

To obtain a high simulation speedup, the sampling technique must not only reduce the number of instructions simulated in the detailed mode but must also reduce the time needed to move the simulation through skipped intervals to the beginning of the next sample to simulate. Thus, a technique to construct the system image (or context) at the beginning of the detailed simulation mode is needed.

In our case, the system image includes both the architectural state (called also sample starting image) and the micro-architectural state (called also sample warm-up). The architectural state is represented by the data values stored in the registers and in the shared memory. These data represent the application contexts and are independent from the processor configuration. The micro-architectural state encompasses cache and branch predictor table contents [20, 4]. As the ARM7 processor cores that we used during experiments implements "not-taken branch prediction", states relative to branch predictors are not included in the checkpoints and the micro-architectural state is represented only by the cache contents.

Both of these states can be constructed either by fast-forwarding the simulation or by checkpointing. Fast-for-

warding simulation aims at reaching a point of interest in the application's instruction stream as fast as possible using simple execution model that ignores most of the architectural details of the system. This kind of simulation helps in constructing a correct program state at the point of interest, while the micro-architectural state may requires some additional warm-up. Generally, the speed of fast-forwarding (or functional simulation) limits the potential acceleration of a sampling technique. Alternatively, checkpointing tends to minimize the simulation time by eliminating the time spent fast-forwarding [20] the instructions between samples executed in the CABA mode. In the context of simulation acceleration, checkpointing consists in storing the program's image right before the simulation starts. If during simulation a certain image is needed, this image is restored by loading the checkpoint.

Checkpoints are generated once for all the DSE by simulating the whole applications. Traditionally, checkpoints for application intervals need to be stored on disk. Thus, a large number of intervals could be prohibitively costly in terms of disk space. In contrast, applying the checkpointing technique in conjunction with our multi-granularity approach reduces the required disk space.

Analyzing multi-granularity phase matrix of an application reveals a lot of similar rows. This similarity can be exploited by storing representative checkpoints instead of storing all the possible checkpoints for all the intervals. Since all the phases on the same matrix line have the same starting point, checkpointing can be done at the starting point of the first occurrence of each group of similar rows encountered in the matrix. During simulation, when a cluster has to be simulated, the checkpoint associated with the first occurrence of the row is loaded. At this point, both the architectural and the micro-architectural states are restored at that point. Thus, instructions belonging to a phase in the first row of group of similar lines will be used as a representative of similar rows.

For instance, considering application "a" in Figure 2.b, if the simulator is at the start 7 and decide to skip an order-2 granularity of instructions, then it will need to load the system state at start 9. Noting that the classifications of the system state for start 2 and start 9 are the same for all orders of granularity, the simulator can load the start 2 state as an equivalent state of the start 9. Similarly all start point with equivalent classification for all orders of granularity can be represented by only one checkpoint, thus saving the checkpoint storage requirement as will be shown in Section 5.3. Figure 2.b shows (in cylinders) the starting points that require checkpointing, for this particular example.

5. EXPERIMENTAL RESULTS

The accuracy and the efficiency of the proposed multi-granularity sampling approach have been evaluated using five benchmarks taken from the MiBench suite [9]: adpcm, rijndael (noted as rj), fft, gsm, and blowfish (noted as bf). These benchmarks have been chosen because they produce imbalanced long clusters. During the experiments, the number of processor cores varied from 4 to 12. Both the "encode" and the "decode" versions of the five benchmarks have been ported to our MPSoC platform. When the number of processors is greater or equal to 4, the same benchmark/version is executed by more than one processor. The simulation framework that is used during our experiments is the MPARM [3]. This framework is composed of several ARM7 cores connected to shared RAM modules via a standard shared AMBA bus. The simulated ARM7 cores have a 4-way set-associative data cache of 4KB and a direct mapped instruction cache of 8KB. All these components are simulated in MPARM at cycle-accurate and bit-accurate levels (CABA) using SystemC.

In this paper, we define simulation speedup as the ratio of the total number of executed instructions by all the processors in full detailed mode divided by the number of executed instructions by all processors when sampling is used. During the experiments, we used an interval size of 50K instructions as the order-1 granularity. On our simulation platform, a full detailed simulation requires several days. For instance, the full detailed simulation of 12 parallel applications with Rijndael and Blowfish, on 12 processors required about 6 days on a 3GHz Pentium-4 host.

5.1 Simulation acceleration of multi-granularity sampling

In this section, we present the simulation speedup based on the proposed multi-granularity approach and we compare the results with the adaptive sampling approach. Figure 3 gives the simulation acceleration for 5 different benchmark combinations, for 4 to 12 processors using adaptive sampling (AS) method with and multi-granularity sampling (MGS) method. To have a good acceleration *vs.* accuracy tradeoff, TWSS was empirically set to 20%.

Considering four concurrent applications with the multi-granularity approach, the simulation speedup for multigranularity sampling ranges between 14x (for rijndael&gsm) and 52x (for rijndael and blowfish). The average simulation speedup is 28x for all the studied applications. The rijndael&blowfish combinations produce the greatest speedup. This result is due to the almost identical behavior of rijndael and blowfish, with which phases of the same small granularity orders overlap to form MPCs. Additionally, the number of detected phases for rijndael and blowfish is very low (both of them are highly regular). The rijndael&gsm combinations produces the lowest speedup value for two reasons. First, rijndael and gsm have extremely different behaviors; rijndael has high cache miss rate, thus almost 50K rijndael instructions are needed to overlap with 1M gsm instructions in each MPC. The number of instructions executed in each MPC is relativeley large. The low number of the generated MPCs reduces the chance of having similar MPCs. Additionally, when gsm is executed in parallel with rjindael, gsm frequently changes the granularity order to overlap with rijndael due to its irregularity. Figure 4 shows that both versions of gsm (i.e., encode and decode) have seven different granularities (from 10 up to 20). Thus when gsm and rijndael are executed concurrently, in addition of having relatively few MPCs, these MPCs are distinct from each other, thus reducing the speedup.

In contrast, the blowfish&adpcm combinations have a high simulation speed in spite of the different behavior of blowfish and adpcm. The L1 miss rate of blowfish is higher than that of adpcm. So almost 50K instructions of blowfish cover 500K instructions of adpcm in the generated MPCs. This heterogeneity does not impact the simulation speedup because these benchmarks have a periodical behavior and the number of distinct MPCs is small.

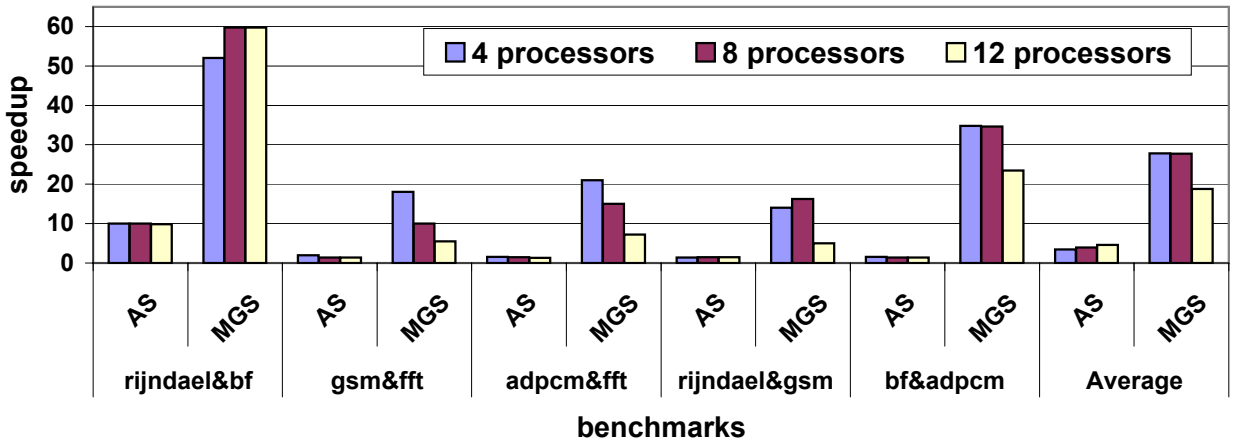

Figure 3: Simulation speedup ratios for 5 combinations of benchmarks with the multi-granularity sampling (MGS) method compared to the adaptive sampling (AS) method. The number of processors varies from 4 to 12 and TWSS=20%.

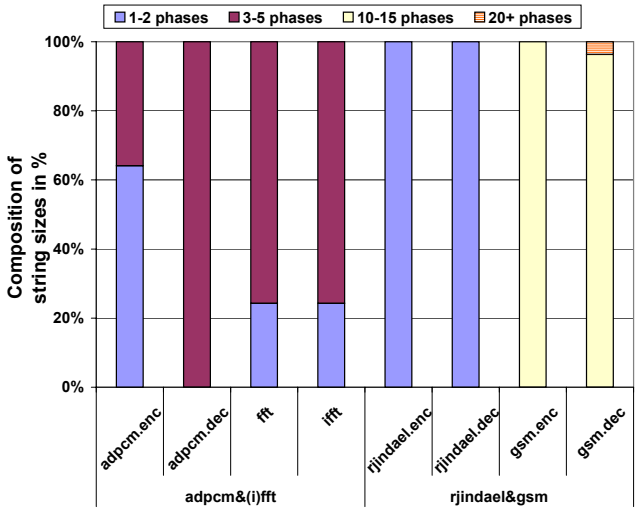

Figure 4: Percentage of the different numbers of phases per string. The benchmark combinations are: adpcm&(i)fft and rijndael&gsm executed on 4 processors with TWSS=20%. All the strings for rijndael encode and decode have one phase.

As shown in Figure 3, the acceleration of the simulation decreases as the number of processors increases. Increasing the number of processes simulated concurrently reduces the simulation speedup significantly for some applications, such as gsm&fft, from 18x for 4 processes to 5x for 12 processes. The decrease in acceleration is due to the decrease in the number of repeated MPCs.

For Adaptive Sampling approach, the obtained speedup depends on the number of phases per string. The smaller the number of phases per string is, the higher the probability of having repeated strings and the greater the simulation acceleration. The combinations of rijndael&blowfish produces the greatest acceleration (about 10x) for two reasons. First, the number of detected phases is small. Second, the identical behavior of rijndael and blowfish generates small phase string in each generated CS. Thus, many CSs are generated, which increases the probability of repeated CS. The acceleration factor of the other benchmark combinations is close to 1.5x (Figure 3). This low acceleration for adaptive sampling is due to the difficulty that this approach faces in detecting similarity of string of phases overlaps.

Figure 4 shows the number of phases in the generated strings when 2 applications are executed in parallel on 4 processors using the adaptive sampling. In this figure, the parallel applications are, on one side, adpcm with fft and, on the other side, rijndael with gsm. As shown, multiple gsm.decode phases (up to 20) are generally needed to cover only one rijndael encode or decode phase. This length disparity between applications reduces the recurrences of the same CS in adaptive sampling. In addition, Figure 4 shows the number of different generated string lengths for each processor, which is quite small. For instance, while executing adpcm and fft in parallel, two string lengths are detected for adpcm.enc (62% of the generated strings contain 2 phases and 38% contain between 3 and 5 phases).

It is clear that the adaptive sampling approach is inefficient in the case of heterogeneous parallel applications. In this situation, long phase strings are generated, thus reducing the number of similar phases strings. Our multi-granularity sampling approach circumvents this problem by analyzing different possible granularities of the samples offline to allow fast and accurate matching of phases during simulation.

5.2 Performance estimation accuracy with multi-granularity sampling

In multi-granularity sampling approach, the IPC estimation error has two sources. The first source corresponds to the added waiting cycles that are injected into the different processors at simulation discretization points. These cycles lower the IPC because no instructions are committed during the wait period. The second source of error is the association of the IPC of recurring MPCs with the IPC of the first simulated MPC. This is due to the fact that intervals classi-

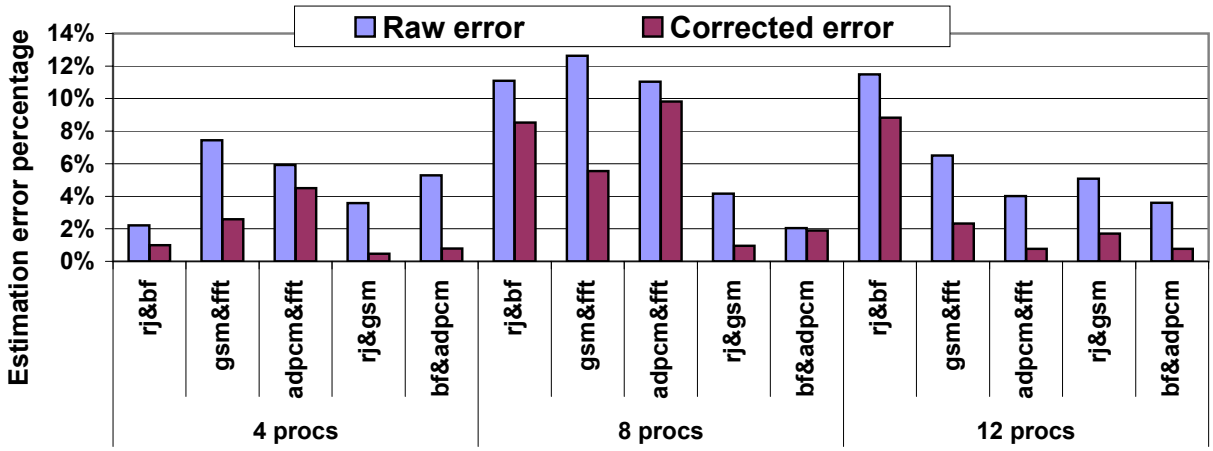

Figure 5: Estimated IPC error for 15 combinations. The number of processors varies from 4 to 12. The TWSS is set at 20%.

fied as similar (same phase) may not have exactly the same performance.

However, injecting the waiting cycles causes the IPC to be underestimated. In order to reduce the error due to the first source, we propose a corrected IPC (Formula 1), which estimates the IPC by eliminating the average waiting cycles, denoted $\text{Avr}_{wait-cyc}$, from the total estimated cycles, denoted Tot_{cycles}. The average waiting cycles, denoted $\text{Avr}_{wait-cyc}$, is estimated using Formula 2. The estimated waiting cycles denoted $\text{Est}_{wait-cyc_i}$ in Formula 2 corresponds to the number of cycles injected during the simulation of a given MPC. The $Freq_i$ corresponds to the frequency of the MPC, Nb_{proc} corresponds to the number of processors and Nb_{MPC} corresponds to the total number of uniquely generated MPCs.

$$Cor_{IPC} = Tot_{instr}/(Tot_{cycles} - Avr_{wait-cyc}) \quad (1)$$

$$Avr_{wait-cyc} = (\sum_{i=1}^{Nb_{MPC}} Est_{wait-cyc_i} * Freq_i)/Nb_{proc} \quad (2)$$

Figure 5 shows both the IPC raw estimation error and the corrected IPC estimation error for the benchmark combinations in Figure 3. This figure demonstrates that the estimated IPC error is smaller than 13% (6% on average) while the corrected IPC error is smaller than 10% (3% on average), as shown in Figure 5. For some benchmark combinations (such as bf&adpcm) the correction can reduce the estimation error by up to 90%. Power consumption estimation error, not shown in this work, is smaller than the IPC estimation error.

5.3 Checkpointing storage and multi-granularity sampling

In this section, we quantify the amount of saving in checkpointing storage when multi-granularity sampling is used. We consider two checkpoints at two starting points of the program as similar if phases for all possible granularities at these start points are classified by simpoint as identical. As detailed earlier in Section 4, in this case the two corresponding rows in the multi-granularity phase matrix look identical.

Figure 6 shows the number of checkpoints when the number of the granularity orders increases from 5 to 20. For the base case in Figure 6, row similarity is not exploited and checkpoints for all phase starts are taken. Thus, the base case corresponds to the situation where AS is applied. In general, the number of checkpoints increases slightly as the number of the granularity orders increases in the phase matrix, *i.e.*, when there is a disparity between the concurrent applications in term of IPC. In this case, the number of similar rows in the phase matrix decreases. We can conclude that the number of checkpoints needed to be stored is much smaller than the base case. The number of checkpoints is reduced by 99.13% when the concurrent applications requires a granularity orders of up to 20. This value is the largest observed granularity for MiBench application combinations.

We conducted several experiments (using tools such as perfmon) in order to compare the fast-forwarding time overheads to checkpointing time overheads. Intuitively, checkpointing is more profitable when a large number of intervals must be skipped between two consecutive simulated clusters and when the architecture state (especially memory) is minimized. This feature makes checkpointing attractive for embedded system DSE. In order to reduce even more checkpoint sizes, we adopted the two techniques proposed by Biesbrouck et al. [4].

Using the Memory Hierarchy Sate (MHS) technique, cache contents are created from the largest reference cache (64KB in our case), and applying the Touched Memory Image (TMI) allowed reducing the maximal size of the architectural state to 550KB. As the number of checkpoints per application is smaller than 100, we need a maximum of 55 MB to store all the checkpoints. The advent of fewer checkpoints allows holding all of them on the physical memory instead of retrieving them from the hard-drive. Using the checkpointing technique, we measured that 10 million host cycles are needed to load one checkpoint, in order to build the architectural and micro-architectural states per processor. Assuming that the fast-forwarding technique is 100x faster than the detailed simulation, fast-forwarding 50K instructions will need approximately 61 million host cycles in our simulation environment. Consequently, skipping phases (even with

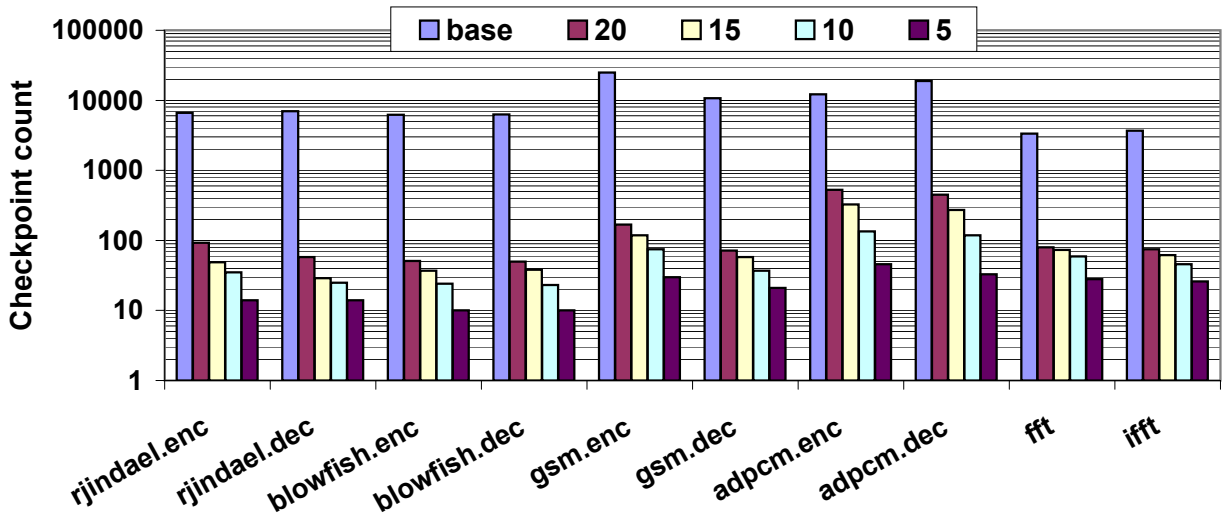

Figure 6: Variation of the number of checkpoints vs. the number of granularity orders. The number of granularity orders varies from 5 to 20. The base corresponds to the number of checkpoints that must be stored if row similarity, in multigranularity phase matrix, is not considered.

small interval size of 50K instructions) with checkpointing is much faster than fast-forwarding them.

6. CONCLUSIONS

In this paper, we present a novel analysis approach of the application that allows efficient prediction of the performance of simulating concurrent heterogeneous applications. We analyze each application behavior for multiple granularities of sampling offline and we create a table of exhibited similarity. At runtime, we form discrete overlaps of phases from the applications running concurrently. We simulate in details only the overlaps that did not show up earlier in simulation.

The simulation speedup, based on our approach reaches 60x. The performance estimation error of our approach is less than 10%, which is acceptable to a wide range of applications. We also devised a corrective formula for the estimated IPC based on our observation of the bias of our technique to underestimate the performance. The proposed IPC correction formula can decrease the error by up to 90%. We showed a technique to reduce the disk storage needed for checkpointing when used in conjunction with our multigranularity approach.

An important advantage of our approach is that the simulation acceleration adapts to the heterogeneous applications interactions without the intervention of the system designer, while maintaining accuracy of performance predictions. Our approach can be applied to a wide class of DSE in embedded systems to reduce the time to market.

Acknowledgment

The experiments presented in this paper were carried out using the Grid'5000 experimental platform, an initiative of the French Ministry of Research, acting through the ACI GRID incentive plan, INRIA, CNRS and RENATER and other contributing partners (see https://www.grid5000.fr).

7. REFERENCES

[1] G. Ascia, V. Catania, A. D. Nuovo, M. Palesi, and D. Patti. Efficient Design Space Exploration for Application Specific Systems-on-a-Chip. *Journal of Systems Architecture*, 53(10):733–750, 2007.

[2] R. B. Atitallah, S. Niar, S. Meftali, and J. Dekeyser. An MPSoC Performance Estimation Framework Using Transaction Level Modeling. *The IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 75–80, Aug. 2007.

[3] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. MPARM: Exploring the Multi-Processor SoC Design Space with SystemC. *Journal of VLSI Signal Processing*, 41(2):169–182, 2005.

[4] M. V. Biesbrouck, B. Calder, and L. Eeckhout. Efficient Sampling Startup for SimPoint. *IEEE Micro*, 26(4):32–42, 2006.

[5] M. V. Biesbroucky, L. Eeckhout, and B. Calder. Considering All Starting Points for Simultaneous Multithreading Simulation. *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 143– 153, Mar. 2006.

[6] M. V. Biesbroucky, T. Sherwood, and B. Calder. A Co-Phase Matrix to Guide Simultaneous Multithreading Simulation. *The IEEE International Symposium On Performance Analysis of Systems and Software (ISPASS)*, pages 45–56, Mar. 2004.

[7] A. Donlin. Transaction Level Modeling: Flows and Use Models. *The ACM International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 75–80, Sep. 2004.

[8] T. Givargis, F. Vahid, and J. Henkel. System-level Exploration for Pareto-optimal Configurations in Parameterized System-on-a-chip. *The IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 416–422, Nov. 2001.

[9] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. *The 4th IEEE Annual Workshop on Workload Characterization (WWC)*, pages 3–14, Dec. 2001.

[10] T. Karkhanis and J. E. Smith. A First-Order Superscalar Processor Model. *Th 31st annual International Symposium on Computer Architecture (ISCA)*, pages 338–349, Jun. 2004.

[11] J. Kihm and D. Connors. CoGS-Sim - CoPhase-Guided Small-Sample Simulation of Multithreaded and Multicore Architecutres. *The 3rd annual Workshop on Modeling, Benchmarking and Simulation (MOBS)*, Jun. 2007.

[12] J. Kihm and D. Connors. PGSS-SIM: Phase-Guided, Small-Sample Simulation. *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 84–93, Apr. 2007.

[13] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder. Motivation for Variable Length Intervals and Hierarchical Phase Behavior. *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2005.

[14] J. Namkung, I. Kozintsev, and C. Dulong. Phase Guided Sampling for Efficient Parallel Application Simulation. *The 4th International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS)*, pages 135–146, Oct. 2006.

[15] S. Nussbaum and J. E. Smith. Statistical Simulation of Symmetric Multiprocessor Systems. *The 35th annual Simulation Symposium (SS)*, pages 89–97, Apr. 2002.

[16] J. Schnerr, O. Bringmann, and W. Rosenstiel. Cycle Accurate Binary Translation for Simulation Acceleration in Rapid Prototyping of SoCs. *The Design, Automation and Test in Europe (DATE)*, pages 792–797, Mar. 2005.

[17] T. Sherwood, E. Perelman, and G. Hamerly. Automatically Characterizing Large Scale Program Behavior. *The 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 45–57, Oct. 2002.

[18] M. Tawk, K. Z. Ibrahim, and S. Niar. Adaptive Sampling for Efficient MPSoC Architecture Simulation. *The 15th of the IEEE international Symposium on Modeling Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Oct. 2007.

[19] P. G. D. Valle, D. Atienza, I. Magan, J. G. Flores, J. M. M. E. A. Perez, L. Benini, and G. D. Micheli. Architectural Exploration of MPSoC Designs Based on an FPGA Emulation Framework. *The XXII Conference on Design of Circuits and Integrated Systems (DCIS)*, pages 12–18, Dec. 2006.

[20] T. Wenisch, R. Wunderlich, B. Falsafi, and J. Hoe. Simulation Sampling with Live-Points. *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 2–12, Mar. 2006.

[21] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. Smarts: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling. *The 30th International Symposium on Computer Architecture (ISCA)*, pages 84–97, Jun. 2003.