WSPS: A Workload Segmented Parallel Simulation Methodology to Accelerate Detailed Simulation

Fan Ni[a], Xiang Long[b], Han Wan[c,*], Xiaopeng Gao[d]
[a,b,c,d] State Key Laboratory of Software Development Environment,
School of Computer Science and Technology, Beihang University, Beijing, China
Email: [a]nifan@les.buaa.edu.cn, {[b]long, [c]wanhan, [d]gxp}@buaa.edu.cn

***Abstract*— Software architecture simulators are indispensable tools in modern processor design. According to the granularity of simulation, they can be classified into the *fast* functional simulation and the *slow* detailed one. The detailed simulator takes far longer time than the functional simulator when simulating the same workload. Based on the duration difference of them, we propose a Workload Segmented Parallel Simulation (WSPS) methodology to accelerate the detailed simulation by simulating different segments of the workload concurrently. The results on SPEC2Kint benchmarks show that, when programs are divided into 64 segments, the speedup is about 11.5, with the relative error of CPI and L1 cache hit-rate remaining lower than 1.5% and 0.01%, respectively. Also, the analysis indicates that WSPS-based simulation can achieve even much higher speedup when using more complicated simulation models, and its duration can approach that of the functional simulation with the accuracy remaining acceptable if the workload size is large enough.**

***Index Terms*— parallel simulation, detailed simulation, functional simulation, acceleration, segment**

I. INTRODUCTION

SOFTWARE architecture simulators are indispensable in the design of modern processors and computers. According to the granularity of simulation, software simulators are generally categorized into: 1) functional simulators, which run to reproduce the functional actions of the real hardware; 2) detailed simulators, which run mainly to mimic the inner behaviour of specific components such as pipeline and cache hierarchies. The latter is often used to measure performance metrics such as cache access statistics at the runtime. Compared with the real hardware, software simulators run much slower due to using hundreds of thousands of host instructions to mimic a single hardware operation. Moreover, all components in the real hardware run concurrently and collaborate to carry out some tasks. Unfortunately, the software simulators are often designed as serial programs, and even as parallel ones, the overhead brought in by the synchronization among different components is often intolerable, which even pay off the benefits of the parallel simulation. As declared by some researches, the fastest functional simulators are several orders of magnitude slower than the direct execution of the hardware, and the detailed performance simulators are even one or more orders of magnitude slower than the functional ones. Increasing enhancement of processors' computation ability in the past decade does not improve the situation, while the ever increasing complexity of the applications makes it even worse, which makes accelerating the simulation process an inevitable problem for researchers. On the other hand, the behaviour of the simulator should match that of the target hardware system in certain degree. Therefore, the aim of the study of simulation acceleration is to minimize the duration and maintain the accuracy of the simulation at the same time.

To reduce the simulation time, researchers have proposed a wealth of acceleration approaches. Some of them [1]–[4] use sampling technology to reduce the instructions to be simulated, and some others parallel the simulation tasks on parallel platforms, such as FastMP [5] and WWT II [6]. All these efforts have achieved success in some degree, but are far from perfect. The first one demands sophisticated sampling mechanisms and compensation methods to avoid sampling bias, which make it too cumbersome. And the latter one needs to coordinate all the components of the system which leads to a complicated implementation. Its practical performance is often dramatically pulled down by the synchronization overhead. furthermore, parallel simulation benefits serial workloads poorly.

In this paper, we propose a novel Workload Segmented Parallel Simulation (WSPS) methodology to accelerate the detailed simulation. The approach speeds up the detailed simulation by dividing the task into segments and carrying them out concurrently. In the methodology, the workload is firstly segmented evenly into N parts (termed as simulation segment), and then *N* simulation instances are initialized to carry out the detailed simulation with one for each simulation segment. When all the instances have completed their simulations, the measurements collected by all instances are synthesized together to generate the measurements for the whole workload. This approach can provide an impressive speedup with a low error rate even when no further compensation is adopted.

In general, the contributions of the paper are twofold: 1) a novel easy-to-deploy simulation methodology is

*Wan Han: Corresponding author, Email: wanhan@buaa.edu.cn.

doi:10.4304/jcp.8.7.1704-1714

presented to accelerate the detailed simulation, which provides good performance and accuracy; 2) detailed measurements and discussions are given for both speedup and accuracy, and some opinions presented in previous researches are explained and conformed.

The rest of this paper is organized as follows. Section II discusses the related work. Section III describes the approach and gives a quantitative analysis of the speedup and accuracy. In Section IV, we present the experiments, and provide the simulation results and analysis. The paper concludes in section V with a summary of our work and future directions.

II. Related Works

Researchers have focus on accelerating detailed simulation for a long time. And a wealth of simulation acceleration technologies are proposed, in which simulation sampling and parallel simulation are the two typical ones.

A. Simulation Sampling

To shorten the duration of the detailed simulation, researchers [7] [8] use abbreviated instruction execution streams of benchmarks as representative workloads. In this approach, the first few hundred million instructions are ”fast-forwarded” to avoid the impact of the initialization phase, and then the following several hundred million or a billion instructions are simulated in detail. This approach only works when the selected instruction stream can represent the characteristics of the complete benchmark. In the first few years of this century, many researchers presented performance results got from abbreviated runs. However, researches like [9]–[14] have concluded that the results obtained from this approach may be inaccurate or even misleading design decision since the behaviour of the abbreviated run may fail to capture the global variations of the benchmark.

Another approach runs the simulation with smaller input sets (as in SPEC2K, test or train sets rather than reference sets are used) to reduce the simulation time. But its drawback is obvious: the behaviour of the benchmarks varies with the input sets. Researches [14] [15] have concluded that the behaviour of the programs varies significantly between test, train and reference sets for most programs in SPEC2K benchmarks.

Researchers propose statistical simulation sampling methodologies to obtain results from complete benchmarks and input sets, the idea behind which is to speculate the benchmarks’ behaviour through simulating selected sections(called *sampling units*). In [9], Thomas M. Conte et al. presented a fast and accurate method for statistical trace sampling which is termed as state-reduction method. SMARTS [4] described a sound procedure to construct minimal sampling sets from full-length benchmarks to enable fast and accurate performance measurements. In the method, functional fast-forwarding and warming-up procedures are used to keep the correct program states. Although functional warming-up enables accurate performance estimation, it limits SMARTS’s speed. The checkpoint technique is often used as an alternative to avoid the slowdown, which yet needs too much space to keep the program states. To overcome this, Wenisch et al. [16] proposed live-points as a replacement for functional warming to reduce simulation turnaround time without sacrificing accuracy and too much storage space. Ref. [17] proposed SimPoint, which uses off-line sampling method. It intelligently chooses the simulation points with a given performance weight by hot-spot analysis and generates the performance results by calculating the simulation points with weight.

The sampling set is large and scattered at all phases of the program execution; it thus usually represents the characteristics of the benchmarks closely. Compared with the abbreviated run, statistical simulation sampling can provide better accuracy; nonetheless, it requires more efforts, including program characteristics analysis and sampling ahead of the deployment (for static sampling) or just-in-time (for dynamic sampling). Furthermore, it needs after-treatment to produce accuracy results.

B. Parallel Simulation

As multi-core and multi-processor platforms become commonplace, parallel simulation have made a big progress. According to parallel simulation, components or simulation tasks are simulated concurrently by exploiting the inherent parallelism of the platforms [18]. Mukherjee et al. [6] identified key simulator operations and tried to minimize their dependence on the host. They proposed a portable, parallel, discrete-event, and direct-execution simulator–WWT II. Matthew C. Chidister and Alan D. George [19] presented a distributed simulator for target CMP platforms based on the Message Passing Interface (MPI). Barr et al. [20] modified the ASIM infrastructure to parallel it. Penry et al. [21] discussed how to automate simulation parallelization and integrate the hardware into CMP models. In FastMP [5], a novel multi-core simulation method was presented, whose speedup is achieved by exploiting the characteristics of the homogeneous multi-processor workloads such as SPECRate. FastMP simulated a subset of cores in detail and the others functionally to reduce the simulation workload. It is clear that, parallel simulation is to accelerate the parts of the simulation which can run concurrently by exploiting the parallelism of the host machine, with serial simulation tasks parallelized (if possible) in certain way.

III. WSPS Methodology

A. Methodology Proposal

Generally speaking, detailed (or timing) simulation is at least one order of magnitude slower than the functional one, and the performance gap can rise up to three or more orders of magnitude. Simulators used by hardware vendors such as Intel Corp. for architecture design are often much slower when the RTL level details are simulated. Fig. 1 presents the detailed and the functional

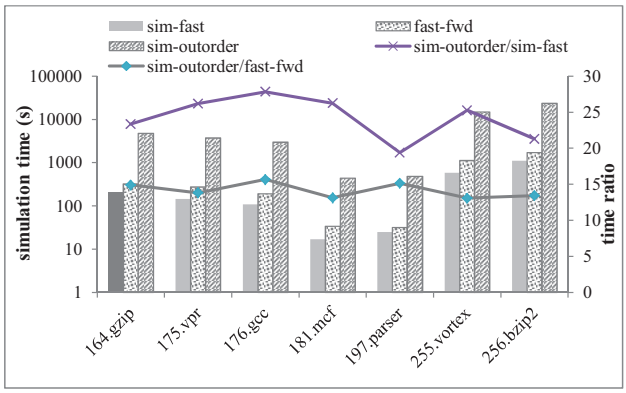

Figure 1. Simulation time: functional vs detailed

simulation time for some typical benchmarks when using Simplescalar [1] toolkit. In the measurements, *sim-outorder* are used for detailed simulation, and *sim-fast* and *fast-fwd* for functional simulation. The *fast-fwd* is the functional *fast-forwarding* model in *sim-outorder* and is slightly more complex than *sim-fast* since it includes an additional simulation loop. In Fig. 1, the upper curve depicts the ratios between the durations of *sim-outorder* and *sim-fast* for the tested benchmarks, which are almost all above 20, with the highest one being 27.84. Similarly, the lower curve depicts the ratios between the durations of *sim-outorder* and *fast-fwd*, which are bigger than 13. Obviously, the detailed simulations last at least one order of magnitude longer than the corresponding functional simulation. Moreover, as our measurements are extraordinarily conservative, the ratio can be reach several hundred when using more detailed timing models.SMARTS [4] argued that the ratio was 60.

As we know, the measurements of performance metrics and the time spent on detailed simulation are scattered through the program execution; therefore the simulation time can be shortened if the simulation loads are divided into segments, which are measured concurrently and finally synthesized to generate the measurements of the whole program. To maintain the accuracy of the measurements at the same time, the programmer-visible architectural components must be in the correct states when each segment's simulation begins. As in simulation based on sampling, these states can be maintained with functional *fast-forwarding* or *checkpoint*. As the duration ratio between the functional and the detailed simulation is usually large as indicated above, the time spent on the functional *fast-forwarding* before the measurements is negligible, and speedup is thus expectable. Moreover, the approach can provide good and manageable accuracy as the bias only exists in the *cold-start* phase in the measurements of each segment. According to this, we propose the Workload Segmented Parallel Simulation (WSPS) methodology to accelerate the detailed simulation. The procedure to deploy the WSPS methodology is as follows:

1) Before the simulation, evenly divide the workload into *N* segments (*0, 1, 2,..., N-1*), with each segment including almost the same number of dynamic instructions to simulate in detail.
2) Concurrently initialize *N* simulation instances numbered [*0,1,2,...,N-1*], and each is assigned one segment of measurements. Only the instructions in the segment are simulated in detail with performance metrics measured, and all instructions before the segment in the execution stream are simulated functionally to keep the correct programmer-visible architectural states. For instance, in simulation instance [*i*], segment [*i*] is simulated in detail and all the dynamic instructions in segment [0,...,*i-1*] are simulated functionally) (see Fig. 2).
3) Synthesize the statistics of all *N* instances to generate the measurements of the whole program. Different performance metrics are handled in different ways. For example, the duration of the simulation equals the maximum of the durations of the *N* instances, and the total simulation cycles, the cache accesses and hits equal the sum of the corresponding counts in all *N* instances, respectively.

B. Speedup Analysis of WSPS

We give a quantitative analysis of the speedup characteristics of the WSPS methodology for homogeneous simulation loads. To make it clear, we assume that the duration of the functional simulation equals *1* while the detailed simulation is *T* (i.e., the ratio of the duration between the two simulations is *T*). According to Fig. 2, the duration of WSPS-based simulation (termed as T_s) equals the maximum of the durations of all N segments. And the duration of the segment [*i*] equals the time spent on the 'fast-forwarding' of the segment [0,...,*i-1*] plus that spent on the detailed simulation of segment [*i*], where $i \in [0, N)$. As the program is homogeneous, the time cost by the detailed and the functional simulation for each segment is T/N and $1/N$, respectively. The instance numbered $N-1$ thus has the largest duration as it needs to 'fast-forward' $N-1$ segments of instructions and then simulate the last segment in detail. Moreover, We can then conclude that $T_s = T_{N-1} = (N-1+T)/N$. The speedup (termed as S) is defined as the ratio between the duration of the normal detailed simulation (T) and that of

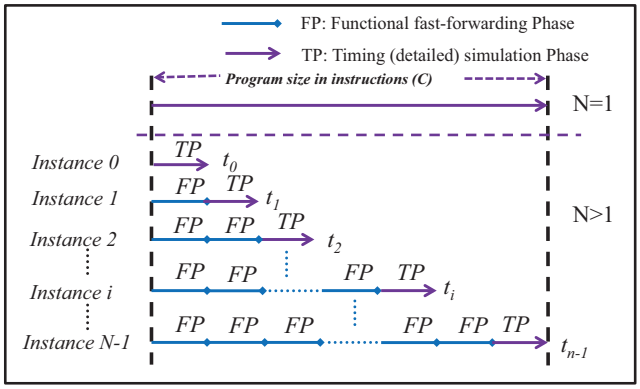

Figure 2. Procedure of WSPS-based simulation

WSPS-based simulation (T_s), that is,

$$S = \frac{T}{T_s} = \frac{N \times T}{N - 1 + T} \tag{1}$$

Equation (1) reveals that the speedup of WSPS-based simulation is determined by the duration ratio *T* and the segment number *N*, and the speedup S approaches *T* as *N* approaches $+\infty$. Fig. 3 depicts the relationship between *N* and *S* when *T* equals 16 and 32, respectively. And the following conclusions can be drawn:

1) Given a *T*, the speedup *S* grows with the increase of the segment number *N*;
2) Given a *N*, the speedup *S* grows with the increase of *T*, and a bigger *N* leads to more significant influence on *S* when *T* varies.

Given a program, quantity of instructions in each segment decreases when the segment number *N* increases. Moreover, when *N* is above the threshold determined by the accuracy, the segment is too small and the effect of the *cold-start* becomes non-negligible which introduces a significant error. In a word, the maximum segment number available to deploy the simulation for a workload is determined by the accuracy required, and a higher accuracy results in a lower segment number, and vice versa.

C. Accuracy Analysis of WSPS

Compared with the statistical sampling simulation, WSPS-based simulation introduces only *cold-start* bias without any sampling bias mentioned in [9]. Now we take the hit-rate of L1 data cache (termed as dL1) as an example to quantitatively analyse the bias introduced by the *cold-start* phase of each segment.

The error of the measurement of the dL1 hit-rate is expressed with relative error (RE), which is defined as

$$RE(Hr) = \frac{Hr - Hr^{'}}{Hr} \tag{2}$$

where Hr is the real hit-rate of the program's dL1 access and $Hr^{'}$ is the hit-rate measured with the WSPS methodology. Accompanied with the fact the bias of the total accesses to the dL1 introduced by *cold-start* is below

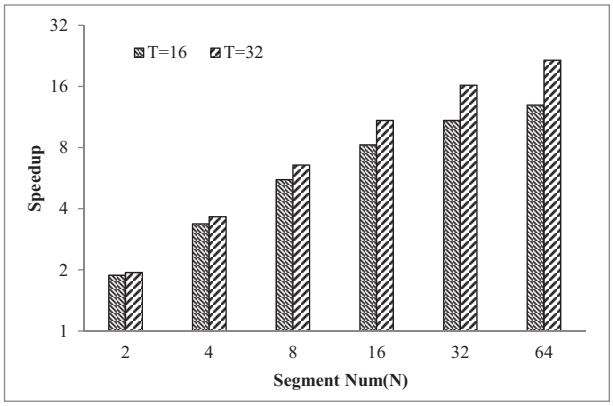

Figure 3. The relationship between *S* and *N* when $T = 16, 32$

1% and the definition of hit-rate, the relative error of the hit-rate of dL1 access is represented as the ratio between the total hits measured and the real ones, that is,

$$RE(Hr) \approx \frac{\Delta(N_h)}{N_h} \tag{3}$$

where N_h is the real total hits of dL1, and $\Delta(N_h)$ is the bias of the measurement of N_h, which equals the total bias introduced by measurements of all *N* segments.

For homogeneous loads, hit-rates of all segments are identical, and the frequency of load/store instructions (termed as *F*) is constant, which means there is one load/store in every *F* dynamic instructions. Therefore, the total hits of dL1 access in each segment equals $N_s \times Hr/F$, where N_s is the number of instructions included in each segment. Then (3) is transformed as follows,

$$RE(Hr) = \frac{\sum_{i=0}^{n-1} \Delta(N_h^i) \times F}{n \times N_s \times Hr} \tag{4}$$

And the relative error of the hit-rate can be further expressed as follows,

$$\begin{aligned} RE(Hr) &= \frac{1}{n} \sum_{i=0}^{n-1} RE(Hr_i^{'}) \\ &\leq max(Hr_0^{'}, Hr_1^{'}, ..., Hr_{n-1}^{'}) \end{aligned} \tag{5}$$

As indicated by (5), the relative error of the program's hit-rate is no more than the maximum of those of the segments. Therefore, if the relative error of the measurement in each segment satisfies the desired accuracy, the accuracy of the complete measurement will also be satisfied.

With a given accuracy, the bigger the bias introduced by the *cold-start* is, the larger the segment size is demanded. The largest bias arises when, in the normal running, all blocks in all sets of the cache are at least accessed once in current segment and contain valid data at the moment when the measurement of current segment begins. In this case, the deviation of hits is $S \times A$, where *S* is the set count of the cache ,and *A* is the association.

Based on the analysis above, the hit-rate of the dL1 access for segment numbered i equals the ratio of the variation and the real hits of this segment, that is,

$$RE(Hr_i) = \frac{S \times A}{N_s \times Hr/F} \tag{6}$$

And from (6) , the minimal size of the segment numbered i satisfying a given accuracy is defined as

$$N_s = \frac{F \times S \times A}{RE(Hr_i) \times Hr} \tag{7}$$

When *F*, *S*, *A* and $RE(Hr_i)$ are set to be 5, 128, 8 and 10^{-5} respectively, and *Hr* is no less than 99%, which is a reasonable assumption for L1 Cache for most applications, (7) indicates that N_s is about $5 * 10^8$. For L1 instruction cache(termed as iL1), it is accessed every time the instruction fetch occurs, thus $F = 1$. And it is clear from (7) that, for identical accuracy, dL1 demands a bigger segment size and determines the minimal segment size

available. As told by some researches [15], the average dynamic instruction size of the programs in SPEC92int, SPEC95int and SPEC2Kint is 1.3, 64 and several hundred billion, respectively. Therefore, if WSPS-based simulation is performed on these benchmarks, the segment number available is 260 and above with an error rate of 10^{-3}. If *T* equals 16, the speedup of WSPS-base simulation above the tranditional deployment will be impressively no less than 15. Moreover, we can see from (1), with a given *T*, the promotion of speedup becomes non-significant when the segment number *N* is big enough, which implies that WSPS-based simulaton can achieve a good speedup with a reasonable segment number.

D. Effectiveness of WSPS for Real Programs

Our discussions on speedup and accuracy of WSPS above are based on the assumption that the load is homogeneous in both execution time and distributions of performance metrics. In the real world, absolutely homogeneous programs are rare if ever exist; therefore, the measurements sometimes deviate from the ideal ones. As shown above, the duration of WSPS-based simulation mostly equals the duration of the last simulation instance. Therefore, if the running time of the benchmark is mostly spent at the tail of the execution, the speedup of WSPS-based simulation will not be so significant. Fortunately, as claimed by the former proposal [22], most programs in the real world spend their execution time mostly on the execution of the tasks, which lies in the middle of the program execution. Hence, when the WSPS methodology is deployed on these programs, the simulation of the instance determining the duration of the whole program overlaps with other instances' functional 'fast-forwarding', and thus the total simulation time decreases. That means, the WSPS methodology can benefit most programs. Discussions presented in Sect. III B can be used to predict to what degree the timing simulation can be speeded up and to what degree we can trust the prediction is explained in Sect. III C.

It is noted that, if we treat N_s and Hr in (7) as the size and hit-rate respectively of a specific segment under consideration, it also holds. Also, the F value is usually nearly unchanged even for heterogeneous program, so for a given accuracy, that is, $RE(Hr_i)$ can be viewed as a constant, only Hr affects N_s in (7). In this case, multiplying a factor (for example 10) to the numerator of (7) will be enough to cover almost all possible changes of Hr in heterogeneous programs. And with the fact that programs in reality are usually large enough, it is easy to draw the conclusion that a much (may be hundred or thousand times) larger segment size than that deduced by (7) is usually available to achieve a given speedup and maintain sufficient accuracy. Therefore, WSPS can be viewed as an perfect choice to speedup timing simulation of both large homogeneous and heterogeneous programs.

IV. Evaluation

To verify the practicability of the methodology and measure the actual speedup of WSPS-based simulation, we deploy the WSPS methodology on plenty of benchmarks and then measure the typical performance metrics.

A. Hardware and Software Environment

We evaluate the WSPS methodology in the context of a multi-issue out-of-order simulator called SegAcc, which includes a functional 'fast-forwarding' model and is developed based on the *sim-outorder* model in SimpleScalar 3.0 [1]. The hardware and software used in the experiment are listed in Table I . It is noted that when the segment number *N* exceeds the cores on the host (N_h), at most N_h instances are initialized to simulate concurrently at a time, termed as a warp, and segments are simulated warp after warp until all of them are completed. This deployment avoids the limitations of the hardware, which implies that the WSPS methodology can be easily deployed on a wide range of platforms, from low-end PCs to clusters.

The detailed simulation model used here is based on *sim-outorder* with some modifications as show in Table II.

As indicated by (1), the speedup of WSPS-based simulation depends on the time ratio, which is determined by the complexity of the timing model. In our deployment, however, we do not complicate deliberately the time model to achieve a better speedup. Therefore, the speedup we display here is quite conservative, and the actual speedup can be more striking with a more detailed timing model.

B. Benchmarks

In the measurements, we use SPEC2Kint and Coremark benchmarks to evaluate our WSPS methodology.

TABLE I.
HARDWARE AND SOFTWARE CONFIGURATION

Hardware	Software
processor: Intel Quad Q6600, 2.4GHz (i/d)L1 Cache: 64KB, 8-way uL2 Cache: 8MB, 8 way Memory: 4G DDR2	OS: Ubuntu 8.10 the Kernel: 2.6.28 Compiler: Gcc-4.3 Compile Option: -O0

TABLE II.
CONFIGURATIONS OF THE DETAILED SIMULATION MODE

Target ISA simulated	PISA (MIPS-like ISA)
The pipeline	6 stages(fetch, dispatch, issue, refresh, writeback, commit)
branch predictor	Bimodal predictor with 2048 entries
Cache(dL1/iL1)	128-set, 8-way, 32-byte block, LRU, Write-back
Cache(uL2)	2048-set, 8-way, 32-byte block, LRU, Write-back
Memory latencies	18 cycles for the first chunk and 2 cycles between remaining chunks
Other configurations	default values provided by *sim-outorder*, for example, fetch/decode/retire width is 4 and size of register update unit (RUU) is 16

As the cross compiler is too old, several programs from SPEC2Kint do not run correctly on SegAcc, so we leave them to our future improvement and just skip them in current measurements. To evaluate the effect of program size on accuracy, we choose Coremark [23], which is an open source benchmark to measure the performance of CPU used in embedded systems. The reason why we chose it instead of programs from SPEC2Kint is that its program size is easy to scale with different iterations and it is homogeneous in program behaviour. Table III shows some useful program attributes in our measurements. All the programs listed in the table are deployed on SegAcc and the CPI, hit-rates of the (i/d)L1/(u)L2 caches and iTLB/DTLB of each program are measured. To reduce the durations of the measurements, we fed the SPEC2Kint benchmarks with *test* input set, instead of *train* or *reference* set. From the following discussions, we will see that, when the *train* or *reference* input set is used, the methodology can provide even higher accuracy.

C. Speedup Evaluation

As indicated in (1), the ideal speedup S grows with the increase of the segment number N, and approaches T when N is big enough. Fig. 4 depicts the speedup when programs are divided into 2, 4, 8, 16, 32 and 64 segments, respectively. As shown, the speedup measured matches the trend of the ideal one when N varies, which can be concluded when comparing the left curves with their counterparts on the right side. When considering the values of T in Table III and the speedups revealed in Fig. 4, it is easy to conclude that with a given N, a bigger T can produce a bigger speedup (S) as revealed by (1). For example, *175.gcc* and *197.parser* have bigger T (15.66 and 16.10, respectively) than the other five benchmarks, so their speedups (S) are also bigger.

Our measurements also show that the duration of WSPS-based simulation equals that of the instance numbered *N-1* in almost all cases, and the detailed simulation is in fact at least one order of magnitude slower than the functional one. We can then conclude that the detailed simulation of the last segment can often determine the duration and the assumption we made for (1) is hence reasonable in most cases.

TABLE III.
ATTRIBUTES OF THE BENCHMARKS

Program name	Instruction size	T
164.gzip	4351546993	14.92
175.vpr	3227211462	13.80
176.gcc	2410588278	15.66
181.mcf	405724024	13.12
197.parser	387542103	16.10
255.vortex	12491133639	13.09
256.bzip2	23661497915	13.44
Coremark	From 1546523446(1K-ieration) to 5154877237(10k-iteration)	approximate 9

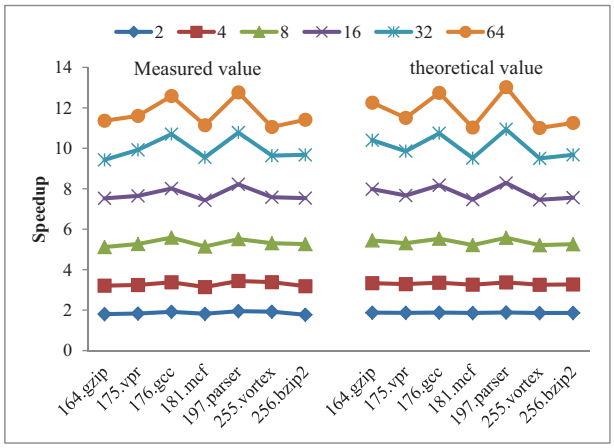

Figure 4. speedup vs N: Measured(left) and theoretical(right)

D. Accuracy Evaluation

We have just discussed the speedup of WSPS-based simulation, and we will go further to discuss the relationship between accuracy and segment number, segment size and program size, respectively.

The accuracy of the measurement of metric V is evaluated quantitatively with *relative error* ($RE(V)$), which can be defined as:

$$RE(V) = \frac{V - V_t}{V} \tag{8}$$

where V is the real value, and V_t is the one measured. **Accuracy vs Segment Number.** Fig. 5 depicts the *RE*s of CPI and hit-rates of dL1, iL1, uL2, iTLB when different segment numbers are used. In all measurements, they are no more than 1.4%, $6.38{*}10^{-5}$, 0.02%, 28.7%, $2.1{*}10^{-6}$ and $1.52 * 10^{-5}$, respectively. It also reveals that, for a given metric such as the CPI, the relative error grows with the increase of the segment number N. Two factors can account for this. Firstly, in WSPS-based simulation, errors are only brought in by the *cold-start* phase of the detailed simulation of every segment, and the *cold-start* bias grows when the segment number N increases, as a bigger N leads to a smaller segment size and thus fewer instructions amortize the *cold-start* effect in each segment. Secondly, a bigger segment number means that more *cold-start* phases are included in the final measurements, and the accuracy is thus decreased as indicted in (4).

Fig. 5 also tells that the patterns of the uL2 access and hit vary greatly for different benchmarks. The relative error rates of uL2 hit-rate for most programs are no more than 6%, expect for *197.parser*, which is 28.8%. The difference in the pattern of L2 cache access accounts for this. Accesses to uL2 in *197.parser* are rarer, about one in every 341 instructions and the hit-rate of uL2 cache is also low, about 68%. For programs like *197.parser*, the significant error of the uL2 measurements comes from: 1) the *cold-start* bias of the access to uL2 cache in detailed simulation of every segment, which is not compensated by further accesses in the same segment, either because the segment is too small or the access

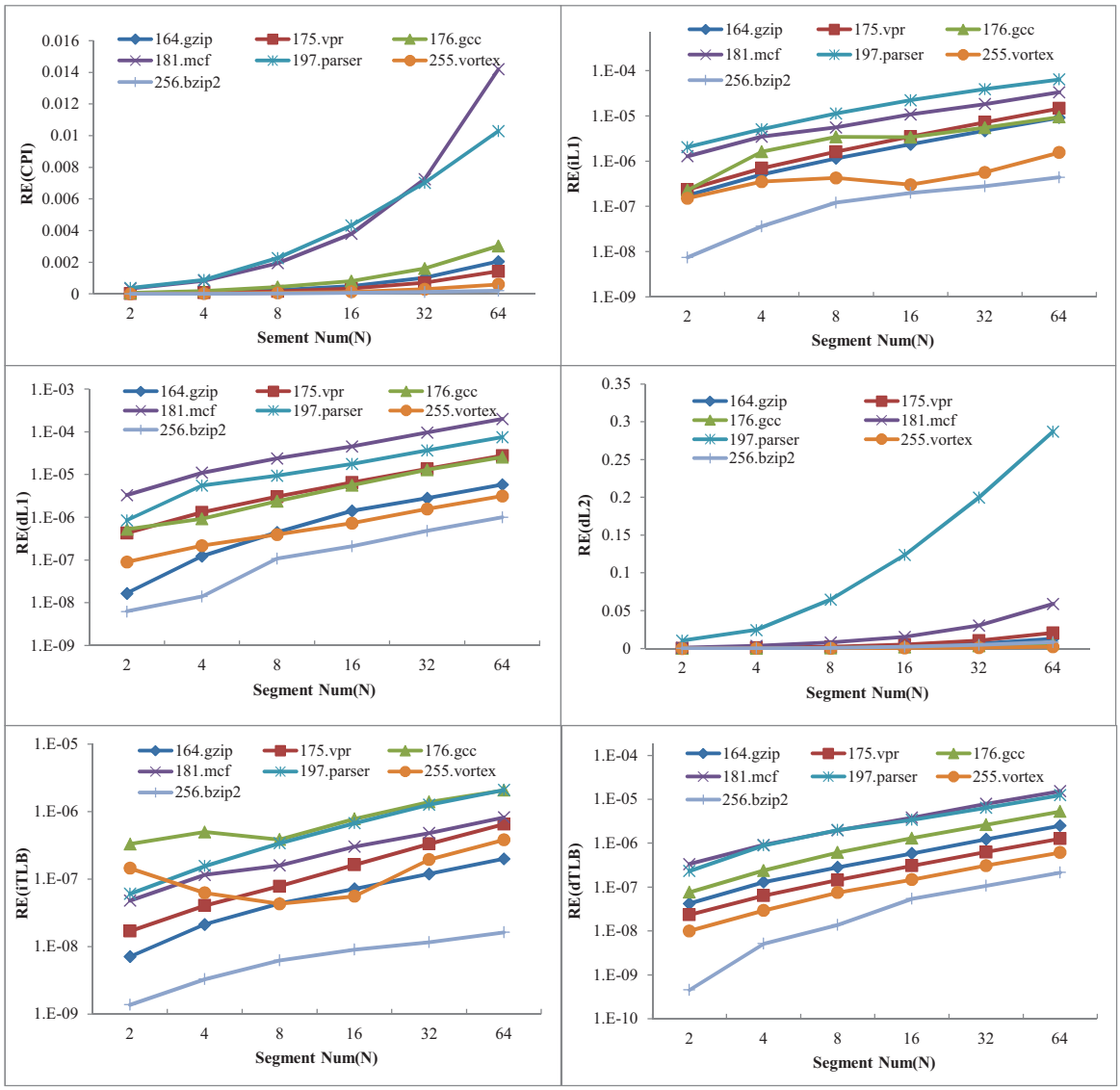

Figure 5. Relative Error vs Segment Num

frequency is too low; 2) the low hit-rate, which enlarges the relative error as indicated by (6). Our studies also show that, with functional warming during functional fast-forwarding and compensation strategies such as treating the first access to uL2 cache as hit, the bias of L2 cache related measurements can be decreased significantly when the error is already large.

The reasons why the error of the uL2 measurements is much more significant than other measurements such as CPI and hit-rates of the L1 cache are explained as follows:

1) Based on intuitive judgments, as the L2 cache is accessed much rarer than the L1 cache, which is only about one in every 100 iL1 cache accesses, its hit-rate will be more sensitive to the variation of the total hits than the L1 cache as indicated by the definition of hit-rate. And we confirm this in (6), which indicates that a much larger set size S (usually ten times larger than dL1 cache), a higher access frequency F (more than 100 times larger than iL1 access) and a lower hit-rate of L2 cache can lead to a lower accuracy with the identical segment number N.
2) In the cache model used in the measurements, every miss and write-back of the L1 cache causes a L2 cache access. In WSPS-based simulation, every miss of the L1 cache access in the *cold-start* phase of the detailed simulation of every segment leads to a L2 cache miss which may be avoidable in the traditional simulation as the L1 access hits. And biases of both iL1 and dL1 measurements are brought into the uL2 measurements as separated L1 and unified L2 cache are adopted in our deployment. Moreover, the access of L2 cache is much rarer than that of L1 cache. Therefore, the segment number N affects the hit-rate more significantly.

From our measurements and analyses, it can be concluded that WSPS-based simulation shows excellent performance in both speedup and accuracy. And our study

also shows that, for programs with few L2 cache accesses and hits, further compensations are required to get more accurate L2 cache measurements.

Accuracy vs Segment Size. We now discuss the relationship between accuracy and segment size. As revealed in (6), the accuracy grows with the increase of the segment size. Our measurements also follow the trend, as shown in Fig. 6. We also compare the results with the theoretical values calculated by (6), and it shows that the practical error rate is always below the corresponding ideal value, and the error rate decreases with the growth of the segment size. Therefore, we can draw the conclusion that the error in our measurements is controllable, and can be limited according to the value as calculated by (6).

It is noted that, the results of CPI, hit-rates of dL1, uL2 and dTLB of *164.gzip* seem anomalous in Fig. 6. This can be explained as follows: when the segment number is fixed to be 16 and the segment size is small, only a small piece of code is simulated. For *164.gzip*, this piece of code may only include the initialization phase, which acts quite differently in the way to access the data cache and data TLB, so the dL1 and dTLB measurements are not consistent. Moreover, the pattern of the dL1 access affects the uL2 cache and the simulation cycles, which leads to abnormal results in the measurements of the hit-rate of uL2 and CPI. However, the iL1 and iTLB measurements are not quite affected by the program's data access pattern, so they follow the trend. We believe that, when the segment size is big enough to cover sufficient code other than the initialization phase, the measurements will also follow the trend. We conform this by extra measurements as shown in Fig.7. It is clear that, when the segment size is bigger than 0.5 million, all measurements of *164.gzip* also follow the trend.

Accuracy vs Program Size. Fig. 8 pictures the accuracy trend of *Coremark* when the program size grows and the segment number equals 8. Some curves at the bottom are enlarged on the top right for clarity. We do not give the datum of uL2 as the access is too rare and the hit-rate is almost 0. The relative error rates of the measurements decrease when the program size increases. It can be explained as follows: when the segment number is fixed and the program size grows, the segment size grows in proportion, and the effect of the *cold-start* bias in the measurements decreases, which accounts for a lower error rate.

Combined with the aforementioned discussions of the relationship between accuracy and segment number and size, Fig. 8 also implies that for a given accuracy, when the program size grows, the segment number available for methodology deployment grows, and thus the speedup increases. Now we can say, the WSPS methodology has a very good scalability.

Impact of large cache on accuracy. According to the discussions above, it is clear that the error brought in by WSPS mainly lies in the *cold-start* of the detailed simulation phase. In the measurements above, we assume a 512K unified L2 cache, which may be too conservative for mainstream machines used today in reality and lead to an underestimation of the errors brought in by the warming-up of a bigger cache. To make the results more convincing and representative, we also deployed the WSPS methodology on a platform with larger LLC, where the unified L2 cache is set to 2-Mbyte. According to the discussions above, the error of the WSPS methodology grows when the segment number becomes larger. So we only run the experiments with N being 64 and the results are listed in Table IV, in which the results with small L2 cache (512K) are also presented. We here only show the results of the two benchmarks (*255.vortex* and *256.bzip2*). It is clear that the errors of CPI and uL2 hit-rates become more significant when a larger L2 cache is used. This is due to the fact that more instructions are needed to warm up the cache and fewer instructions left in the detailed simulation phase to offset the bias brought in by the *warming-up*. Even though the error grows, it is acceptable, and the relative error of CPI and uL2 hit-rates are no more than 0.2% and 3.5%, respectively.

Moreover, with the fact that the accuracy of WSPS grows with the increase of the program size and the benchmarks we use here are relative small, the accuracy can be much higher when WSPS is deployed on real programs or benchmarks with larger input sets, such as *reference* input set of SPEC2K.

To sum up, the WSPS mechanism can indeed accelerate the detailed simulation of large benchmarks and provide accurate simulation results in modern processor design and evaluation.

E. Comparison with Existing Work

In [4], the author claims that the functional simulation model they used is about 60 times faster than the detailed simulation, under which Smarts provides a speedup of about 35 over detailed simulation for 8-way out-of-order processors and the average error of CPI is about 0.64%. [17] puts focus mostly on explaining how to select *representative* simulation points off-line to make the results of the sample simulation more believable and the experiment results show that the average error of CPI is about 2.1% when multiple simulation points are used.

According to the discussions in Sect. III, with the same functional and detailed simulation models, i.e., $T = 60$, the ideal speedup provided by the WSPS methodology can be as high as 31.2 when N equals 64. Although we do not carry out the simulation on the same models as used in [4] and [17] due to lack of information to represent,

TABLE IV.
ERRORS BROUGHT IN BY SMALL AND LARGE L2 CACHE

Relative Error	255.vortex (512K/2M)	256.bzip2 (512K/2M)
CPI	5.96E-4 / 1.39E-3	2.00E-4 / 9.90E-4
iL1	1.55E-6 / 1.58E-6	4.38E-7 / 4.38E-7
dL1	3.14E-6 / 3.13E-6	9.97E-7 / 1.05E-6
uL2	2.26E-3 / 6.03E-3	8.44E-3 / 3.40E-2
iTLB	3.82E-7 / 6.04E-7	1.62E-8 / 1.62E-8
dTLB	6.12E-7 / 6.12E-7	2.13E-7 / 2.13E-7

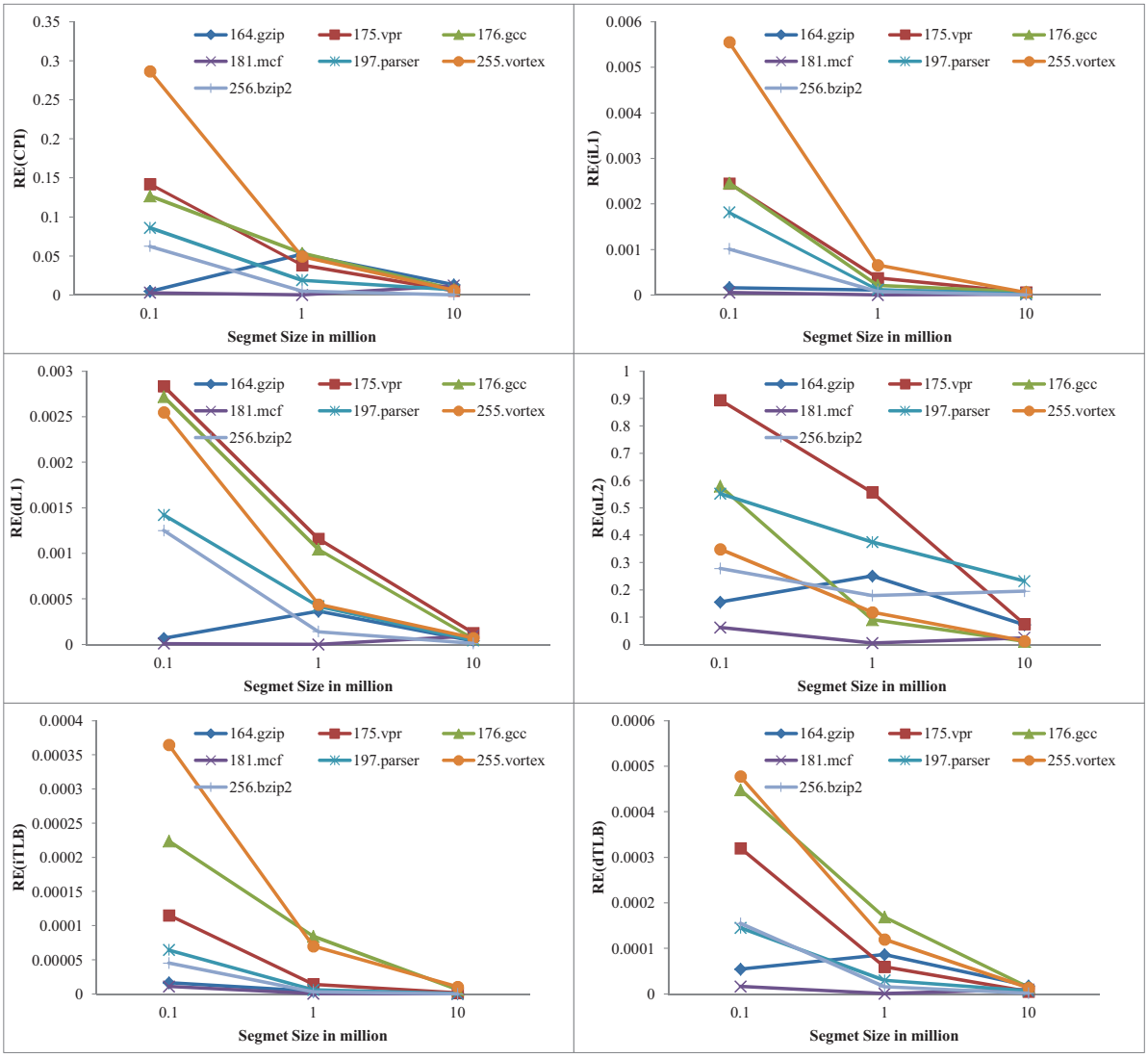

Figure 6. Relative Error vs Segment size when N equals 16

it is reasonable to claim WSPS can provide comparable speedup to Smarts and Simpoint due to the fact that the real speedups match the ideal ones, as shown in Figure 4. Meanwhile, the WSPS mechanism can provide at least as accurate results as sampling-based methods, such as Smarts and Simpoint, due to the fact that it only incurs *cold-start* bias while the sample-based methods also include sampling errors.

Compared to detailed simulation acceleration technologies existing, the WSPS methodology has several main advantages, apart from high speedup and accuracy. Firstly, it is intuitive and simple to deploy. No efforts are needed to design the sampling algorithm and select the *representative* samples. Secondly, it is easier to analyse the accuracy of the simulation due to the fact that only *cold-start* bias exists in the simulation. Thirdly, it is possible to get the statistics (for example CPI or cache performance) of a certain period of the simulation, which is usually not possible in sampling-base simulations unless the period is selected as a *sample*. Lastly, the performance of our mechanism scales perfectly with the size of the program, and thus it is particularly suitable to be deployed on large programs which tend to cost a long time to simulate.

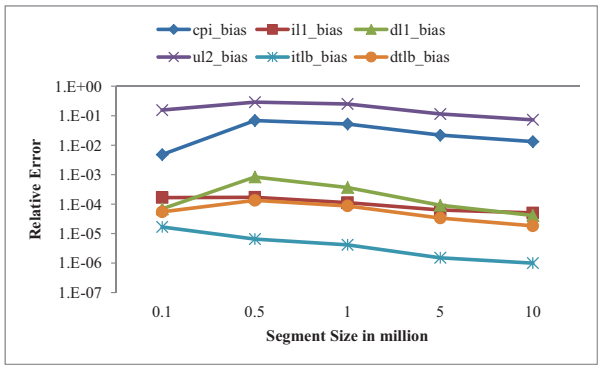

Figure 7. Accuracy vs Segment Size for 164.gzip

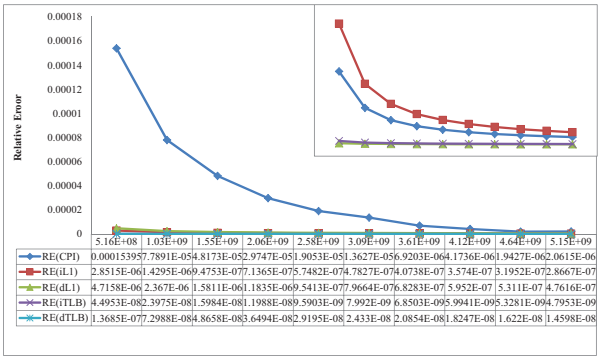

Figure 8. Relative Error vs Program size

F. Brief Summary

With the above discussions, some conclusions are drawn. Firstly, WSPS-based simulation can provide a good speedup, which depends on the segment number and the detailed-functional simulation time ratio. Secondly, the actual speedup does not exactly equal the ideal one but follows its trend. Thirdly, for most benchmarks, the methodology can provide sufficient accuracy, and for loads like *197.parser*, it provides good measurements for most performance metrics except the L2 cache related ones, which can be remedied by other technologies such as component *warming-up*. Lastly, the accuracy of the measurements decreases as the segment number increases for a given load, and increases as the program size grows for the same load with a given segment number.

V. Conclusion

In this paper, we propose a novel methodology for detailed simulation to reduce the simulation duration. As shown in our measurements, the WSPS methodology can provide a sound speedup while maintain acceptable accuracy at the same time. Compared with traditional acceleration methods, the WSPS methodology is suitable to accelerate detailed simulation due to the following advantages: 1) high accuracy, as the complete measurements are synthesized by complete execution of the benchmark with a number of simulation instances, and no *sampling bias* thus exists; 2) good scalability of speedup, the speedup grows with segment number, which can be bigger with the increase of benchmark's program size for a given accuracy, and the execution time can approach that of the functional simulation when the segment number is big enough; 3) excellent support to highly detailed time simulation, as the more detailed the time model is, the higher speedup is predictable; 4) convenience to deploy the methodology on a variety of hosts, ranging from clusters to PCs, as the methodology imposes no assumption on the hosts where the methodology is deployed.

The future work includes: 1) developing approaches to accelerate the functional *fast-forwarding*, so as to enlarge the time ratio *T* and thus produce better improvement; 2) introducing *warming-up* phase before the measurements of the performance metrics to enhance accuracy; 3) combining the WSPS methodology with other acceleration technologies such as SMARTS [4] and live-points [16]; 4) giving quantitative analysis of real-world programs' speedup and accuracy; 5) considering the possibility to deploy the WSPS methodology on parallel workloads.

Acknowledgment

This work is supported by the fund of the State Key Laboratory of Software Development Environment in BUAA grant SKLSDE-2009ZX-02 and the Chinese National 863 program grant 2007AA01Z183.

References

[1] D. Burger and T. M. Austin, "The SimpleScalar Tool Set, Version 2.0," Computer Sciences Department, University of Wisconsin-Madison, Technical Report 1342, 1997.

[2] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Heo, "Statistical Sampling of Microarchitecture Simulation," *ACM Transactions on Modeling and Computer Simulation*, vol. 16, no. 3, pp. 197–224, 2006.

[3] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, and J. C. Hoe, "SimFlex: statistical sampling of computer system simulation," *IEEE Micro*, vol. 26, no. 4, pp. 18–31, 2006.

[4] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe, "Smarts: Accelerating microarchitecture simulation via rigorous statistical sampling," in *Proceedings of the International Symposium on Computer Architecture*, 2003.

[5] S. Kanaujia, I. E. Papazian, J. Chamberlain, and J. Baxter, "FastMP: A Multi-core Simulation Methodology," *in MOBS 2006: Workshop on Modeling, Benchmarking and Simulation*, 2006.

[6] S. S. Mukherjee, S. K. Reinhardt, S. K. Reinhardt, M. Litzkow, M. D. Hill, D. A. Wood, S. Huss-Lederman, and J. R. Larus, "Wisconsin wind tunnel ii: A fast, portable parallel architecture simulator," *IEEE Concurrency*, vol. 8, no. 4, pp. 12–20, Oct.-Dec. 2000.

[7] M. J. Charney and T. R. Puzak, "Prefetching and memory system behavior of the spec95 benchmark suite," *IBM Journal of the Research and Development*, vol. 41, no. 3, pp. 265–286, 1997.

[8] A.Sodani and G.S.Sohi, "An empirical analysis of instruction repletion," *ACM SIGPLAN Notices*, vol. 33, no. 11, pp. 35–45, Nov. 1998.

[9] T. Conte, M. Hirsch, and K. Menezes, "Reducing state loss for effective trace sampling of superscalar processors," *International Conference on Computer Design*, pp. 468–477, Oct. 1996.

[10] S. L. M. Rabeb Mizouni, "Simulation-based feature selection for software requirements baseline," *Journal of Software*, vol. 7, no. 7, pp. 1440–1450, July 2012.

[11] T. Lafage and A. Seznec, "Choosing representative slices of program execution for microarchitecture simulations: A preliminary application to the data stream," in *IEEE Workshop on Workload Characterization, ICCD*, Sept. 2000.

[12] L. L. Fachao Jiang, Molin Wang, "Software design of engine characteristic simulation," *Journal of Software*, vol. 7, no. 2, pp. 316–321, Feb. 2012.

[13] G. Lauterbach, "Accelerating architectural simulation by parallel execution of trace samples," in *Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, vol. 1, Jan. 1994, pp. 205 –210.

[14] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[15] W. C. Hsu, H. Chen, and P. C. Yew, "On the predictability of program behavior using different input data sets," in *Proceedings of Sixth Annual Workshop on Interaction between Compilers and Computer Architectures*, Feb. 2002, pp. 45 –53.
[16] T. F. Wenisch, R. E. Wunderlich, B. Falsafi, and J. C. Hoe, "Simulation sampling with live-points," *IEEE International Symposium on Performance Analysis of Systems and Software(ISPASS 06)*, pp. 2–12, 2006.
[17] G. Hamerly, E. Perelman, and B. Calder, "How to use simpoint to pick simulation points," *ACM SIGMETRIC Performance Evaluation Review*, 2004.
[18] T. L. Zhangang Wang, Suping Peng, "Gpu accelerated 2-d staggered-grid finite difference seismic modelling," *Journal of Software*, vol. 6, no. 8, pp. 1554–1561, Aug. 2011.
[19] M. Chidister and A. George, "Parallel simulation of chip multiprocessor architectures," *ACM Transactions on Modeling and Computer Simulation*, vol. 12, no. 3, pp. 176–200, 2002.
[20] K. C. Barr, R. Matas-Navarro, C. Weaver, T. Juan, and J. Emer, "Simulating a chip multiprocessor with a symmetric multiprocessor," in *Boston Area Architecture Workshop*, Jan. 2005.
[21] D. A. Penry, D. Fay, D. Hodgdon, R. Wells, G. Schelle, D. I. August, and D. Connors, "Exploiting parallelism and structure to accelerate the simulation of chip multiprocessors," in *12th International Symposium on High-Performance Computer Architecture*, Feb. 2006, pp. 27–38.
[22] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2001.
[23] EEMBC-Benchmarks, "Coremark: A benchmark that aims to measure the performance of central processing units (CPU) used in embedded systems," http://www.coremark.org/home.php.

Fan Ni is currently a Ph.D. candidate at State Key Laboratory of Software Development Environment, School of Computer Science and Technology, Beihang University, China. He received his BS degree in computer science and technology from Beihang University, China, in 2006.

His research interests include computer architecture simulation technology, operating system and real-time system. During the last few years, he focused on parallel architecture simulation and designed some simulators.

Xiang Long received his MS and PhD degree in computer science and technology from School of Computer Science and Technology from Beihang University in 1988 and 1994, respectively, and his BS degree in mathematics from Peking University 1985.

He is currently a Professor of Computer Architecture at School of Computer Science and Technology, Beihang University, China. He is also Head of Department of Computer Science and Technology at School of Computer Science and Technology. His current research interests include embedded system architecture and software, OS design, and computer system security. During the last few years, he has led a team working on virtualization and got some achievements.

Han Wan She received her PhD degree in computer architecture from Beihang University in 2011.

She is currently working as a Technician at School of Computer Science and Technology, Beihang University, China. Her current research interests include parallel computer architecture, architecture simulation and GPGPU. During the past few years, she keeps working on accelerating cache simulation with CUDA platform and has got some achievements, some of which are published on international conferences.

Xiaopeng Gao He received his PhD degree in computer software and theory from Beihang University in 2002.

He is currently an Assistant Professor of Computer Architecture at School of Computer Science and Technology, Beihang University, China. His current research interests include computer architecture, embedded system and computer network. In the past few years, he has done a lot of work on parallel simulation and GPGPU.