Selecting Benchmark Combinations for the Evaluation of Multicore Throughput

Ricardo A. Velasquez, Pierre Michaud, André Seznec

Selecting Benchmark Combinations for the Evaluation of Multicore Throughput

Ricardo A. Velásquez
INRIA/IRISA
Rennes, France
ricardo-andres.velasquez@inria.fr

Pierre Michaud
INRIA/IRISA
Rennes, France
pierre.michaud@inria.fr

André Seznec
INRIA/IRISA
Rennes, France
andre.seznec@inria.fr

***Abstract*—Most high-performance processors today are able to execute multiple threads of execution simultaneously. Threads share processor resources, like the last-level cache, which may decrease throughput in a non obvious way, depending on threads' characteristics. Computer architects usually study multiprogrammed workloads by considering a set of benchmarks and some combinations of these benchmarks. Because detailed microarchitecture simulators are slow, we want a subset of combinations that is as small as possible, yet representative. However, there is no standard method for selecting such sample, and different authors have used different methods. It is not clear how the choice of a particular sample impacts the conclusions of a study. We propose and compare different sampling methods for defining multiprogrammed workloads for computer architecture studies. We evaluate their effectiveness with a case study, the comparison of several multicore last-level cache replacement policies. We show that random sampling, the simplest method, is a possible way to define a representative workload sample, provided the sample is large enough. We propose a method for estimating the required sample size based on fast approximate simulation. We propose a new method, workload stratification, which is very effective at reducing the sample size in situations where random sampling would require large samples.**

I. Introduction

The performance of an application executing on a multicore processor can be strongly impacted by applications running simultaneously on the other cores, mainly because of resource sharing (last-level cache, memory bandwidth, chip power...). This impact is not obvious, and quantifying it often requires detailed simulations.

The study of multicore performance on multiprogrammed workloads, i.e., sets of independent threads running simultaneously, is still a very active research area. The most widely used method for such study is to use a set of single-thread benchmarks, to define a fixed set of multiprogrammed workloads from these benchmarks, to simulate these workloads and to quantify performance with a throughput metric. The population of possible benchmark combinations may be very large. Hence most studies use a relatively small sample of a few tens, sometimes a few hundreds of workloads. In general, all the benchmarks in a suite are assumed to be equally important. Therefore we would like the sample to be representative of the whole population of possible workloads. Yet, there is no standard method in the computer architecture community for defining multiprogrammed workloads. There are some common practices, but not really a common method. More important, authors rarely demonstrate the representativeness of their workload samples. Indeed, it is difficult to assess the representativeness of workload sample without simulating a larger number of workloads, which is precisely what we want to avoid. Approximate microarchitecture simulation methods that trade accuracy for simulation speed offer a way to solve this dilemma. Approximate simulation is usually advocated for design-space exploration. We show in this study that fast approximate simulation can also help select representative multiprogrammed workloads in situations requiring detailed simulations (e.g., for estimating power consumption).

We investigate several sampling methods, using as a case study a comparison of several multicore last-level cache replacement policies. We performed simulations with Zesto, a detailed microarchitecture simulator [1], and with BADCO, a fast approximate simulator [2]. We show that, unless we know a priori that the microarchitecture being assessed *significantly* outperforms (or underperforms) the baseline microarchitecture, it is not safe to simulate only a few tens of random workloads, as frequently done in many studies. Hence it is necessary to simulate a large workload sample, which is possible with a fast approximate simulator. We propose a method for determining, from a representative subset of all possible workloads, what should be the size of a random workload sample. We propose an improved sampling method, balanced random sampling, that defines workloads in such a way that all the benchmarks are equally weighted in the sample. Sometimes, random sampling requires more than a few tens of workloads. We evaluate an alternative method, benchmark stratification, that defines workloads by first defining benchmark classes. However, this method is not significantly better than random sampling. Finally, we propose a new method, workload stratification, that is very effective at reducing the sample size when random sampling would require too large a sample.

This paper is organized as follows. Section II presents the related work and the current practices. In Section III, we propose a method for obtaining the size of a representative workload sample under random sampling. Section IV describes our experimental setup and briefly presents BADCO, our approximate simulator. We evaluate experimentally our random sampling method in Section V. Then Section VI introduces and evaluates experimentally three alternative sampling methods.

This work was partially supported by the European Research Council Advanced Grant DAL No 267175 and a PhD fellowship funded by Region Bretagne

Section VII gives a practical guideline. Finally, Section VIII concludes this study.

II. Background

Simulation objectives for a computer architect are generally to compare two or more multicore microarchitectures under some criterion such as execution time, multiprogram throughput, power consumption, fairness, etc. Generally one wants also to quantify the differences between microarchitectures. In this study we consider the problem of evaluating multiprogram throughput, i.e., the quantity of work done by the machine in a fixed time when executing simultaneously several independent threads. The usual procedure for evaluating multiprogram throughput is to take a set of benchmarks (e.g., the SPEC CPU benchmarks) and define some combinations of threads executing concurrently, on which the microarchitectures are evaluated. We call *workload* a combination of K benchmarks, K being the number of logical cores[1]. The number of workloads out of B benchmarks is generally very large. If the cores are identical and interchangeable, and assuming that the same benchmark can be replicated several times, there is a population of $\binom{B+K-1}{K}$ possible workloads. Because detailed microarchitecture simulators are very slow, computer architects generally consider a *sample* of W workloads where W is typically only a few tens. The microarchitectures being compared are simulated on all W workloads. For each microarchitecture, we obtain a total of $W \times K$ IPC (instructions per cycle) values, denoted IPC_{wk}, where $w \in [1, W]$ is the workload and $k \in [1, K]$ is the core. The $W \times K$ IPC values are then reduced to a single throughput value via a throughput metric. The microarchitecture whose throughput value on the W workloads is the highest is deemed to be the one offering the highest throughput on the full workload population.

The workload sample is generally much smaller than the full population, but there is no standard method for defining a representative sample, although there are some common practices. Yet, the method used for selecting the sample may change the conclusions of a study dramatically.

A. Current practices

We did a survey of the papers published in three major computer architecture conferences, ISCA, MICRO and HPCA, from 2007 to march 2012. We identified 75 papers that have used fixed multiprogrammed workloads[2]. The vast majority of these 75 papers use a small subset of all possible workloads, ranging from a few workloads to a few hundreds. Many papers use a few tens of workloads and compute an average performance on them. Of the 75 papers, only 9 use a completely random selection of workloads. The 66 other papers classify benchmarks into classes and define workloads from these classes. In the vast majority of cases, the classes are defined "manually", based on the authors' understanding of the problem under study. Then, some workload types are defined. For instance, if there are two benchmark classes A and B and two identical cores, 3 types of workloads may be defined: AA, BB and AB. Then a certain number of workloads are defined for each workload type. The number of workloads and the method for defining them is more or less arbitrary. The practices here are very diverse depending on the author and on the problem studied. For instance, some authors choose to give more weight to certain workload types, sometimes without any reason. Some authors select benchmarks randomly under the constraint of the workload type. Some others choose a single benchmark to be representative of its class.

[1]Physical cores may be SMT

[2]We do not count the studies using a number of benchmarks small enough for allowing to simulate all the possible workloads.

B. Systematic methods

Only a few papers have explored the problem of defining representative multiprogrammed workloads. The most obvious systematic method for defining multiprogrammed workloads is random selection. The advantage of random workload selection is that it is simple and less susceptible to bias. Indeed, if the author of a study has a very good understanding of a problem, he/she can identify "important" workloads. However, the behavior of modern superscalar processors is sometimes quite complex, and detailed simulators are needed to capture unintuitive interactions. This is why research in computer architecture is mostly based on simulation. Defining multiprogrammed workloads a priori, based on one's understanding of the studied problem, may inadvertently bias the conclusions of the study. Though random selection of workloads is a simple and obvious method, it is not clear how many workloads must be considered. Van Craeynest and Eeckhout have shown in a recent study [3] that using only a few tens of random workloads, as seen in many studies, does not permit evaluating accurately a throughput metric like weighted speedup [4] or harmonic mean of speedups [5]. In their experiments, about 150 random 4-thread workloads are necessary for computing throughput with reasonable accuracy from 29 individual SPEC2006 benchmarks. That is, random selection requires a workload sample larger than what is used in most studies. That may justify why most authors prefer to use a class-based selection method instead. Among the studies using class-based workload selection, very few are fully automatic. In a recent study, Vandierendonck and Seznec use cluster analysis to define 4 classes among the SPEC CPU2000 benchmarks [6]. Van Biesbrouck et al. [7] described a fully automatic method to define workloads using microarchitecture-independent profiling data. Instead of classifying benchmarks, they apply cluster analysis directly on workloads.

C. Approximate simulation

Several approximate microarchitecture simulation methods have been proposed [3], [8], [9], [10], [11], [2], [12], [13] (the list is not exhaustive). In general, these methods trade accuracy for simulation speed. Approximate simulation is usually advocated for design space exploration and, more generally, for situations where detailed simulators are too slow.

D. Throughput metrics

Throughput represents the quantity of work done per unit of time. There are several possible ways to define a throughput metric depending on how the unit of work and the total quantity of work are defined. In this study we consider the

three most frequently used throughput metrics : the IPC throughput (**IPCT**), the weighted speedup (**WSU**) [4], and the harmonic mean of speedups (**HSU**) [5]. These throughput metrics can be summarized with a single formula [14]: the *per-workload throughput* $t(w)$ for workload w is

$$t(w) \quad = \quad \underset{k\in[1,K]}{\text{X-mean}} \ \frac{IPC_{wk}}{IPC_{ref}[b_{wk}]} \tag{1}$$

where X-mean is the arithmetic mean (A-mean) or the harmonic mean (H-mean), IPC_{wk} is the IPC of the thread running on core k, $b_{wk} \in [1, B]$ is the benchmark running on core k, and $IPC_{ref}[b]$ is the IPC for benchmark b running on a reference machine. The sample throughput is computed from the W per-workload throughput numbers:

$$T \quad = \quad \underset{w\in[1,W]}{\text{X-mean}} \ t(w) \tag{2}$$

A metric equivalent to the IPCT can be obtained by setting X-mean to A-mean and $IPC_{ref}[b]$ to 1. WSU and HSU are obtained by setting X-mean to A-mean and H-mean respectively; and for $IPC_{ref}[b]$ we use the IPC of the benchmark running alone on the reference machine (*single-thread IPC*).

III. Random sampling

As noted in Section II, random sampling is not the most popular method in the computer architecture community. Many authors prefer to work with a relatively small sample that they try to define (more or less carefully) so that it is representative. Yet, random sampling is a safe way to avoid biases, provided the sample is large enough. Moreover, random sampling lends itself to analytical modeling. We present in the remainder of this section a model for estimating the probability of drawing correct conclusions under random workload selection.

For a fixed W, the sample throughput defined by formula (2) can be viewed as a random variable, the sample space for that variable being all the possible subsets of W workloads out of a full population of $N = \binom{B+K-1}{K}$ workloads. The problem of comparing two microarchitectures X and Y can be stated as follows. We want to know whether or not Y yields a greater throughput than X. Let T_X and T_Y be the sample throughput of microarchitectures X and Y respectively. T_X and T_Y are two random variables. For the IPCT and WSU metrics, we define random variable D:

$$D \quad = \quad T_Y - T_X = \underset{w\in[1,W]}{\text{A-mean}} \ d(w) \tag{3}$$

where the random variable $d(w)$ is defined as

$$d(w) = t_Y(w) - t_X(w) \tag{4}$$

In words, $d(w)$ and D are respectively the per-workload and per-sample throughput difference. If we have some information about the distribution of D, we may be able to compute the probability that D is positive. Because the W workloads are chosen randomly and independently from each other, the Central Limit Theorem (CLT) applies, and D can be approximated by a normal distribution [15].

Let μ and σ^2 be respectively the mean and variance of $d(w)$. The mean of D is also equal to μ and its variance is σ^2/W, assuming $W \ll N$. The **degree of confidence** that Y is better than X is equal to the probability that D is positive:

$$Pr(D \geq 0) \quad = \quad \frac{1}{2}\Big[1 + \text{erf}\Big(\frac{1}{c_v}\sqrt{\frac{W}{2}}\Big)\Big] \tag{5}$$

where $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ is the error function and $c_v = \sigma/\mu$ is the coefficient of variation of $d(w)$.

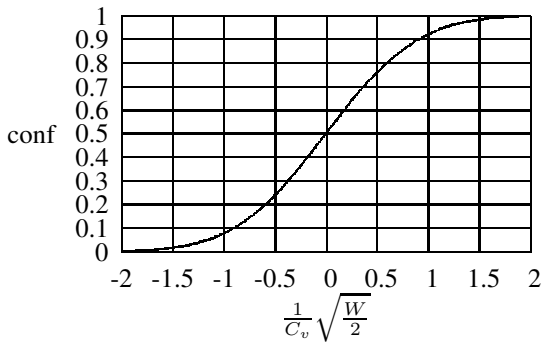

Fig. 1. Degree of confidence as a function of $\frac{1}{c_v}\sqrt{\frac{W}{2}}$ (equation (5).

For the HSU metric, a H-mean is used in formulas (1) and (2), and it is the inverse of the HSU on which the CLT applies[3]. Thus for the HSU we define the random variable D as

$$D \quad = \quad \frac{1}{T_X} - \frac{1}{T_Y} = \underset{w\in[1,W]}{\text{A-mean}} \ d(w) \tag{6}$$

with the random variable $d(w)$ defined as

$$d(w) = \frac{1}{t_X(w)} - \frac{1}{t_Y(w)} \tag{7}$$

that is, $d(w)$ and D are respectively the per-workload and per-sample reciprocal throughput difference. The coefficient of variation c_v of $d(w)$ is used in equation (5).

Figure 1 shows the degree of confidence as a function of $\frac{1}{c_v}\sqrt{\frac{W}{2}}$ (equation (5)). A degree of confidence close to zero means that it is very likely that Y is *not* better than X. The degree of confidence becomes very close to 0 or 1 for

$$\left|\frac{1}{c_v}\sqrt{\frac{W}{2}}\right| = 2$$

Solving this equation for W, we obtain the required sample size:

$$W = 8c_v^2 \tag{8}$$

The only parameter needed in this model is the coefficient of variation c_v, which is estimated from experiments. We present an experimental validation of the model in Section V-A.

[3]Our goal is not to discuss which throughput metric should be used or not. The CLT applies to any throughput metric that can be expressed as a sum of per-workload terms. For instance, if one prefers to quantify throughput as a geometric mean of speedups [14], i.e., if a geometric mean is used in formulas (1) and (2), the CLT applies to the logarithm of throughput, which leads to define the random variables $D = \log T_Y - \log T_X$ and $d(w) = \log t_y(w) - \log t_x(w)$.

TABLE I. CORE CONFIGURATION.

decode/issue/commit	4/6/4
RS/LDQ/STQ/ROB	36/36/24/128
DL1/DTLB MSHR entries	16/8
Clock	3 GHz
IL1 cache	2 cycles, 32 kB, 4-way, 64-byte line, LRU, next-line prefetcher
ITLB	2 cycles, 128-entry, 4-way, LRU, 4 kB page
DL1 cache	2 cycles, 32 kB, 8-way, 64-byte line, LRU, write-back, IP-based stride + next line prefetchers
DTLB	2 cycles, 512-entry, 4-way, LRU, 4 kB page
Branch predictor	TAGE 4 kB, BTAC 7.5 kB, indirect branch predictor 2 kB, RAS 16 entries

TABLE II. UNCORE CONFIGURATIONS.

	2 cores	**4 cores**	**8 cores**
LLC size/latency	1MB/5cyc.	2MB/6cyc.	4MB/7cyc.
DL1 write buffer	8 entries		
LLC	64-byte line, 16-way, write-back, 8-entry write buffer, 16 MSHRs, IP-based stride + stream prefetchers		
FSB clock	800 MHz		
FSB width	8 bytes		
DRAM latency	200 cycles		

IV. EXPERIMENTAL EVALUATION

A. Simulation setup

We use two microarchitecture simulators in this study, Zesto and BADCO. Zesto is a detailed x86 simulator [1]. BADCO is an approximate simulator for fast uncore simulation [2]. We present BADCO shortly in section IV-B. Our experiments analyze the performance of symmetric multicores with 2, 4 and 8 identical cores. Table I summarizes the core characteristics. As a case study, we consider five uncore microarchitectures, each uncore corresponding to a different shared last-level cache replacement policy: LRU, RANDOM (RND), FIFO, DIP [16] and DRRIP [17]. Table II gives the uncore characteristics. We build the workloads from 22 of the 29 SPEC CPU2006 benchmarks (the 22 benchmarks that we were able to simulate with Zesto). We simulate every uncore using BADCO for the full population of workloads whenever possible (253 workloads for 2 cores, 12650 workloads for 4 cores), or for a large sample when the number of possible combinations is huge (we consider 10000 workloads for 8 cores). We also perform Zesto simulations for 250 randomly selected workloads for 2, 4 and 8 cores, and for every uncore. We compiled all the benchmarks with gcc-3.4 using the "-O3" optimization flag. For generating BADCO traces, we skip the first 40 billion instructions of each benchmark, and the trace represents the next 100 million instructions (no cache warming is done). We assume that simulations are reproducible, so that traces represent exactly the same sequence of dynamic μops. We used SimpleScalar EIO tracing feature [18], which is included in the Zesto simulation package. During multiprogram execution, each core runs a separate threads. When a thread has finished executing its 100 million instructions earlier than the other threads, it is restarted[4]. This is done as many times as necessary until all the threads in the workloads have executed at least 100 million instructions. The IPC is measured only for the first 100 million committed instructions of each thread.

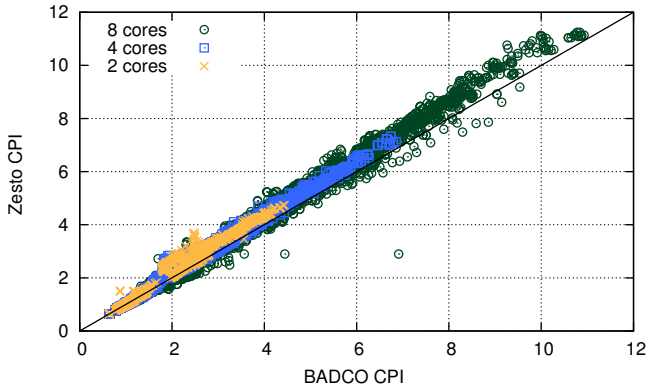

Fig. 2. Zesto CPI (cycles per instruction) vs. BADCO CPI.

B. BADCO

BADCO is a behavioral application-dependent superscalar core model for fast uncore simulation [2]. As a behavioral core model, BADCO emulates the *external* core behavior (i.e. the way the core communicates with the uncore), not the mechanisms inside the core. Such behavioral model is derived from detailed microarchitecture simulations. In particular, BADCO uses two traces to build a core model. A BADCO model is specific to a particular benchmark running on a particular core. A core model is basically a directed graph where nodes represent groups of μops and their associated uncore requests. The graph edges represent inferred dependences between nodes. Once a core model is built, it can be plugged into an uncore simulator to quickly evaluate the performance of several uncore configurations. The trace-driven simulation is performed by a *BADCO machine*. A BADCO machine is an abstract core that fetches and executes *nodes*. A detailed description of BADCO can be found in our previous work [2], along with an evaluation of its single-thread simulation capabilities. In the present work, we use BADCO to simulate multiprogrammed workloads. Extending BADCO to execute multiprogrammed workloads is straightforward. Once BADCO core models have been built for a set of single-thread benchmarks, the core models can be easily combined to simulate a multicore running several independent threads simultaneously. We connect several BADCO machines, one per core, to a detailed uncore simulator[5]. BADCO machines send read and write requests to the uncore. There is a round robin arbitration to decide which BADCO machine can access the uncore. When the uncore receives a request, it translates the virtual address to a physical address (if a page miss occurs, BADCO allocates a new physical page). After that, the uncore

[4]More rigorous multiprogram simulation methods could be used, such as the co-phase matrix method [19]. The problem of defining representative benchmark combinations is orthogonal and concerns the co-phase matrix method as well.

[5]Our uncore simulator was extracted from Zesto.

TABLE III. BADCO AVERAGE SIMULATION SPEEDUP.

Number of cores	1	2	4	8
MIPS - Zesto	0.170	0.096	0.049	0.017
MIPS - BADCO	2.52	2.41	1.89	1.19
Speedup	14.8	25.19	38.88	68.1

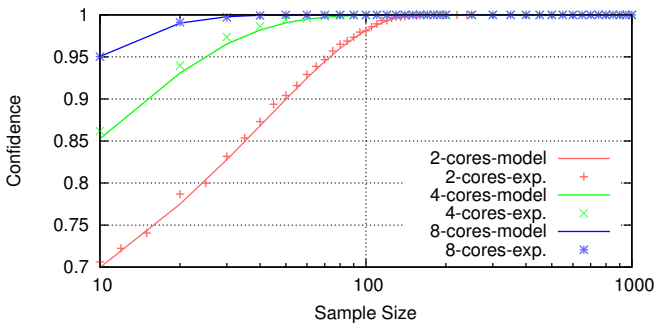

Fig. 3. Degree of confidence that DRRIP outperforms DIP as a function of the sample size (throughput metric: WSU). Experimental measurements vs. analytical model (equation (5)).

processes the request and informs the core when the request has been completed.

Figure 2 shows the CPI (cycles per instruction) for Zesto and BADCO respectively. Each dot represents the CPI of a benchmark in one of 250 benchmark combinations. If BADCO and Zesto were identical, all the dots would lie on the bisector. Here, most points are close to the bisector but above it, that is, BADCO slightly underestimates the CPI. The average CPI error is 4.59 %, 3.98 % and 4.09 % for 2, 4 and 8 cores respectively. In all cases, the maximum error is less than 22 %. For approximate simulators, more important than predicting CPIs accurately is predicting speedups accurately. We compared the speedups predicted by BADCO and Zesto for several replacement policies. We found that, on average, the speedup error is 0.66 % 0.61 % and 1.43 % for 2, 4 and 8 cores respectively. BADCO is notably better at predicting speedups than raw CPIs.

Table III reports the simulation speed of Zesto and BADCO in million instructions simulated per second (MIPS). BADCO is clearly faster than Zesto, the simulation speedup increasing with the number of cores. It should be recalled that BADCO and Zesto use the exact same uncore model.

V. EXPERIMENTAL RESULTS FOR RANDOM SAMPLING

A. Random sampling model validation

We experimentally validated formula (5) for the 10 pairs of replacement policies, for the 3 metrics (IPCT, WSU and HSU) and for 2, 4, and 8 cores. We measured the *experimental degree of confidence* that policy Y outperforms policy X for a given sample size by generating 1000 random samples: the experimental degree of confidence is the fraction of samples for which the sample throughput of Y is greater than that of X. Figure 3 shows the result of this experimental validation for one pair of policies and one metric: the model curve matches the experimental points quite well, even for small samples. Although not shown due to the limited space, the other metrics and policy pairs exhibit similarly good matching between the model and the experiment.

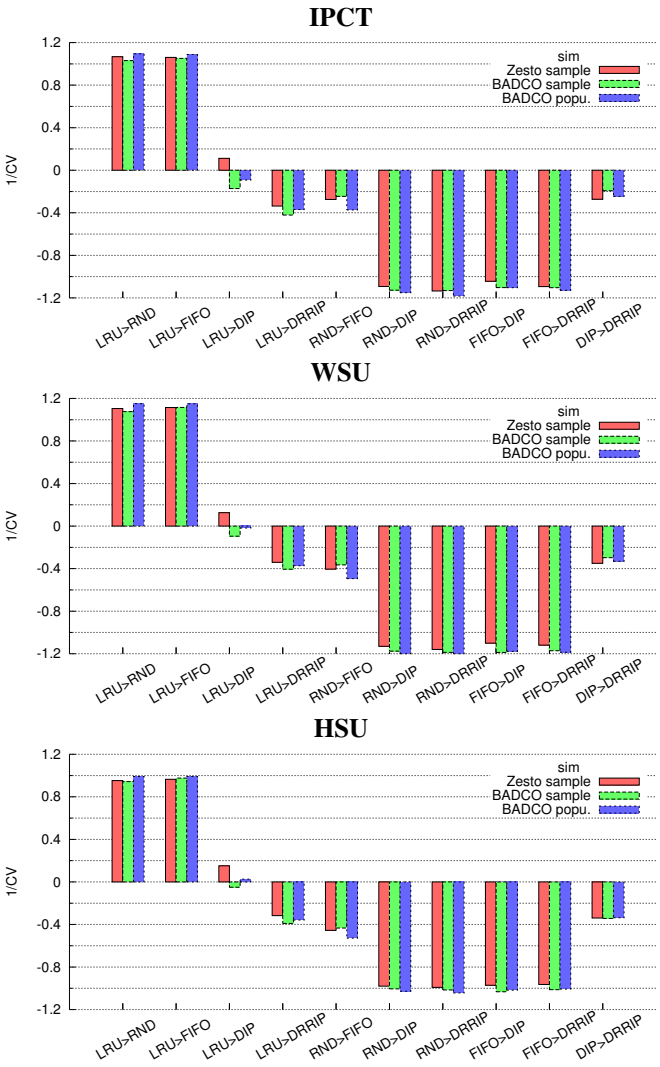

Fig. 4. Inverse of the coefficient of variation ($1/c_v$) for each policy pair and each metric on a 4-core processor. The first bar in a group of 3 gives $1/c_v$ measured with Zesto on a 250-workload sample. The second bar gives $1/c_v$ measured with BADCO on the same 250-workload sample. The third bar gives $1/c_v$ measured with BADCO on the full 12650-workload population.

B. The tinier the performance difference, the more workloads are needed

Random workload selection requires knowing the appropriate sample size, i.e., the number of workloads that we must consider for drawing conclusions consistent with the full workload population with a reasonably high probability. As explained in Section III, the coefficient of variation c_v of the random variable $d(w)$ is the only parameter needed to decide the sample size. Figure 4 shows the inverse of the coefficient of variation ($1/c_v = \mu/\sigma$) for each pair of replacement policies, assuming a 4 core processor. The sign of $1/c_v$ indicates which policy in a pair performs best. The magnitude $|1/c_v|$ gives an indication of the performance difference between the two policies.

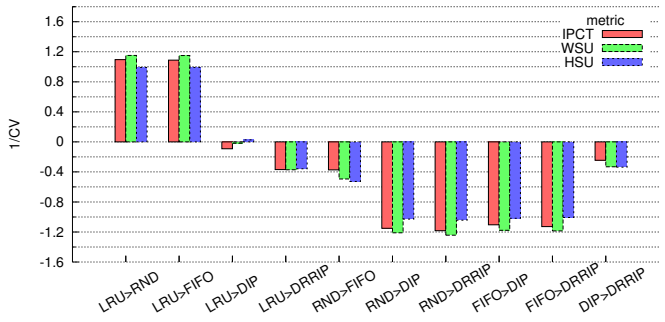

Fig. 5. Inverse of the coefficient of variation ($1/c_v$) measured with BADCO on the full population of 12650 workloads for 4 cores.

When the performance difference between two policies is significant, $|1/c_v|$ is relatively large. For instance, LRU significantly outperforms FIFO on all 3 metrics, and $c_v \approx 1$. From formula (8), about 8 randomly chosen workloads are sufficient to compare LRU and FIFO. In accordance with intuition, the larger the performance difference between two microarchitectures the fewer workloads are necessary to identify the best of the two.

However, when two policies have very close performance, such as LRU and DIP, $|1/c_v|$ is much smaller than 1. In such situation, a reasonable conclusion is that the two policies perform almost equally. However, we need a very large sample even for drawing this conclusion. For instance, the value of $|1/c_v|$ for LRU vs. DIP is smaller when computed on the full population than on the 250-workload sample. We cannot be certain that the value of c_v estimated on a sample is accurate unless we know a priori that one microarchitecture significantly outperforms the other. Two microarchitectures may have the same average performance on the full population of workloads, yet one microarchitecture may seem to outperform the other on a sample. In other words, if we have no a priori reason to believe that one microarchitecture significantly outperforms the other, we must consider a workload sample as large as possible. A fast qualitatively-accurate simulator such as BADCO allows to consider a large workload sample. If the sample is large enough, we can use it to estimate the coefficient of variation c_v. If c_v is greater than 10, we must conclude that the two policies perform equally on average.

C. Different metrics may require different sample sizes

The physical meaning of a throughput metric depends on some assumptions regarding benchmarks and what they represent. Different metrics rely on different assumptions [14]. Computer architecture studies sometimes use several different throughput metrics to show that the conclusions are robust. Figure 5 shows the inverse of the coefficient of variation ($1/c_v$) for different pairs of policies on 4 cores and for the 3 throughput metrics. On this particular example, the sign of c_v does not depend on the throughput metric. That is, all 3 metrics rank replacement policies identically on a large enough workload sample. However, the magnitude of $|c_v|$ is not the same for all metrics. For example, when comparing RND and FIFO, $|1/c_v| \approx 0.4$ using the IPCT and $|1/c_v| \approx 0.5$ using the HSU. It means that a random sample of $8c_v^2 = 32$ workloads is sufficient with the HSU, but it would not be sufficient for the IPCT, which requires a random sample of $8c_v^2 = 50$ workloads. If one wants to use simultaneously several different throughput metrics on a fixed random workload sample, the required sample size must be determined for each metric, and the selected sample must be large enough for all metrics.

VI. Alternative Sampling Methods

A. Balanced Random Sampling

If we consider the full population of workloads and count how many times a given benchmark occurs overall, we find that all the benchmarks occur the same number of times. This is consistent with the implicit assumption that all the benchmarks are equally important. Random sampling, that we have considered so far, assumes that all the workloads have the same probability of being selected and that the same workload might be selected multiple times (though unlikely in a small sample). However, there is no guarantee that all the benchmarks occur exactly the same number of times in such random sample.

We propose another form of random sampling, *Balanced Random Sampling*. Balanced random sampling guarantees that every benchmark has the same number of occurrences in the whole sample. Hence, after picking a workload, all the workloads in the population may not have the same probability of being selected. We have no mathematical model for this kind of sampling. Instead we have drawn 10000 balanced random samples and have computed experimentally the degree of confidence. Figure 6 shows the degree of confidence estimated with BADCO for several different sampling methods, including random sampling and balanced random sampling (the other methods are introduced afterwards). Compared to simple random sampling, balanced random sampling is a more effective method, providing higher confidence for a given sample size. Balanced random sampling is also, on average, the second most effective sampling method. However, there are still some situations such as the one in the top graph of Figure 6 where the required sample size is very large.

B. Stratified Random Sampling

The workload population is generally not homogeneous. For example, let us assume that microarchitecture Y consistently outperforms microarchitecture X on 80 % of the workload population, while X consistently outperforms Y on the remaining 20 %. The knowledge of these subsets allows us to define a more representative sample. Instead of taking a single sample of W random workloads, we could take $0.8 \times W$ samples randomly from the first subset and $0.2 \times W$ workloads randomly from the second subset. This is a well-known method in statistics, called *stratified sampling* [15]. The method generalizes as follows. The full population of N workloads is divided into L subsets $S_1, S_2, ..., S_L$ of $N_1, N_2, ..., N_L$ workloads respectively. The subsets, called *strata*, are non overlapping, and each workload in the population belongs to one stratum, so we have

$$N_1 + N_2 + ... + N_L = N$$

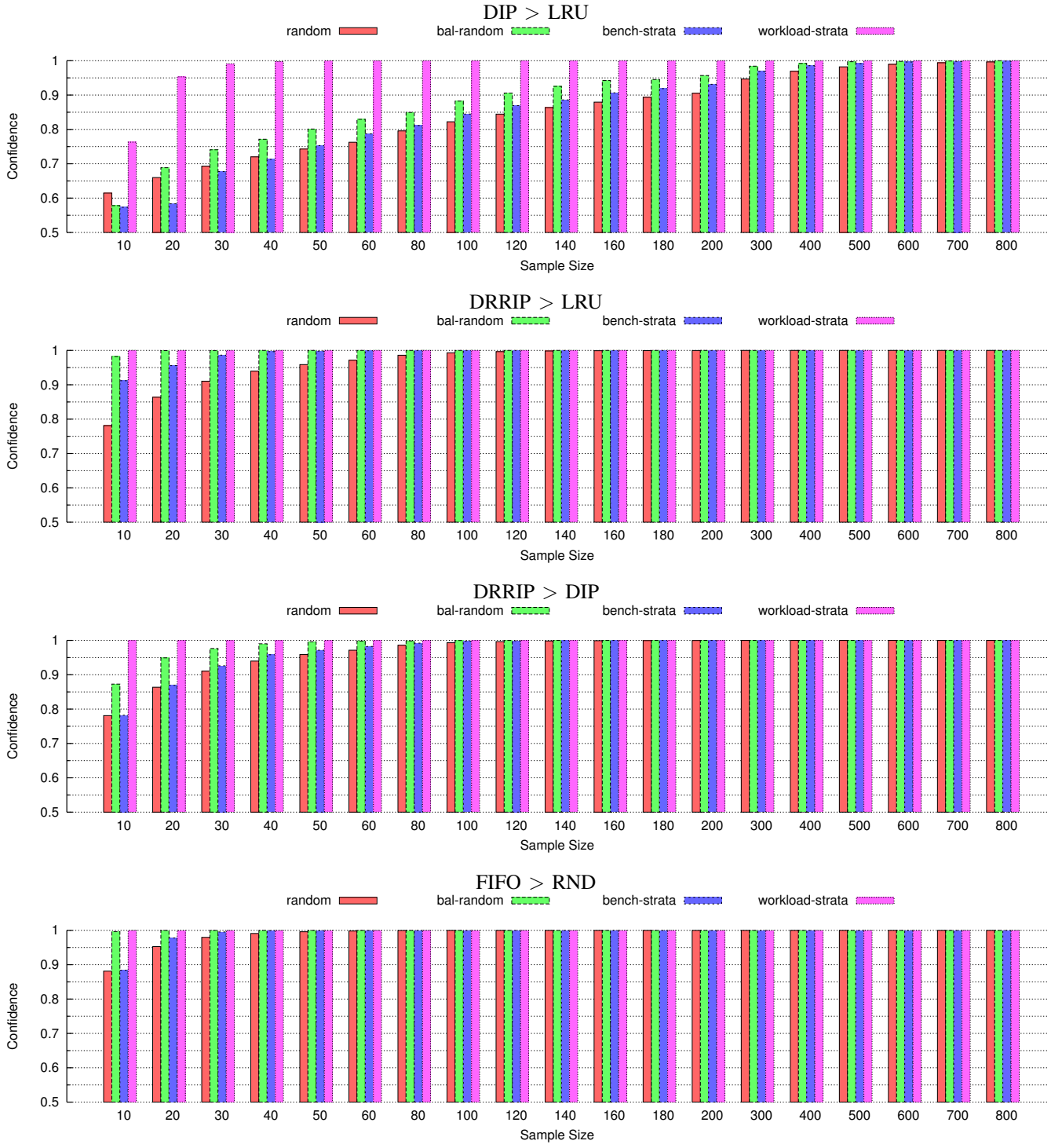

Fig. 6. Experimental degree of confidence estimated with BADCO as a function of the sample size for 4 policy pairs ($DIP > LRU$, $DRRIP > LRU$, $DRRIP > DIP$, $FIFO > RND$), 4 sampling methods (simple random sampling, balanced random sampling, benchmark stratification and workload stratification), assuming 4 cores and using the IPCT metric.

MPKI Class	Benchmarks
Low	povray, gromacs, milc, calculix, namd, dealII, perlbench, gobmk, h264ref, hmmer, sjeng
Medium	bzip2, gcc, astar, zeusmp, cactusADM
High	libquantum, omnetpp, leslie3d, bwaves, mcf, soplex

TABLE IV. CLASSIFICATION OF SPEC BENCHMARKS ACCORDING TO MEMORY INTENSITY: LOW ($MPKI < 1$), MEDIUM ($MPKI < 5$), AND HIGH ($MPKI \geq 5$).

Once strata are defined, a random sample of W_h workloads is drawn independently from each stratum $S_h, h \in [1, L]$. The total sample size W is

$$W = W_1 + W_2 + ... + W_L$$

Global throughput is no longer computed with formula (2) but with a weighted arithmetic mean (WA-mean) or a weighted harmonic mean (WH-mean) depending on the throughput metric:

$$T = \underset{h\in[1,L]}{\text{WX-mean}} \ \underset{w\in S_h}{\text{X-mean}} \ t(w) \quad (9)$$

where WX-mean stands for WA-mean or WH-mean and where the weight for stratum S_h is N_h/N. If the strata are well defined, it is possible to divide a very heterogeneous workload population into strata that are internally homogeneous, so that the coefficient of variation of each stratum is small. As a result, a precise estimate of throughput for a stratum can be obtained from a small sample in that stratum. There are many different ways to define strata. Ideally, we would like to have the minimum number of strata with minimum W_h that produce maximum precision. It is important to note that stratified sampling requires to draw samples from each stratum. Hence W cannot be less than the number of strata. In the remainder of section, we compare two different ways to define strata: *benchmark stratification* and *workload stratification*.

1) Benchmark Stratification: It is common in computer architecture studies to define multiprogrammed workloads by first defining benchmark classes (cf. Section II-A). The main assumption is that benchmarks in the same class exhibit similar behavior. Benchmark classes by themselves do not constitute strata but allow to build workload strata. We can construct strata according to the number of occurrences of each benchmark class in a workload. For example, the workloads composed of benchmarks all belonging to a given class constitute a stratum. Assuming there are M benchmark classes $C_1, C_2, ..., C_M$, we can represent a stratum with an n-tuple $(c_1, c_2, ..., c_M)$ where c_i is the number of occurrences of class C_i in a workload, with the constraint $\sum_{i=1}^{M} c_i = K$, the number of cores. That is, workloads with the same number of occurrences per class belong to the same stratum. This method defines $L = \binom{M+K-1}{K}$ distinct strata. The size of a stratum is

$$N_h = \prod_{i=1}^{M} \binom{b_i + c_i - 1}{c_i}$$

where b_i is the number of benchmarks in class C_i. Table IV shows a classification of the SPEC CPU2006 benchmarks according to the memory intensity measured in misses per kilo-instruction (MPKI). For a 4 core processor, this classification generates 15 strata, hence $(c_{low}, c_{med}, c_{high})$ = (004, 013, 022, 031, 040, 103, 112, 121, 130, 202, 211, 220, 301, 310, 400). Using this stratification, we have drawn 10000 stratified samples and have estimated experimentally the degree of confidence for policy pairs comparisons. Figure 6 shows the degree of confidence with benchmark stratification. For almost all sample sizes, benchmark stratification increases the degree of confidence to some extent, but does not reduce dramatically the sample size required to reach a high degree of confidence.

It should be noted that the benchmark stratification method described here is an attempt to formalize some common practices that are diverse and not always explicit. The studies we are aware of that define multiprogrammed workloads by first defining benchmark classes neither use stratified sampling nor formula (9). Note also that classifying benchmarks according to the MPKI is probably not the best classification for studying replacement policies. Nevertheless, the effectiveness of benchmark stratification strongly depends on the authors' intuition.

2) Workload Stratification: Fast approximate simulators such as BADCO allow to estimate the throughput on large samples of thousands of workloads. Once approximate throughput values have been obtained for all workloads in the large sample, defining strata directly from these values is straightforward. As we seek to compare two microarchitectures according to a certain throughput metric, we can define strata based on the distribution of $d(w)$ for that pair of microarchitectures (see section III). The proposed method is as follows:

1) Measure $d(w)$ for every workload in the large sample.
2) Sort the workloads according to $d(w)$.
3) Process the workloads in ascending order of $d(w)$, putting workloads in the same stratum
4) When the stratum has reached a minimum size W_T and when the standard deviation of the stratum exceeds a certain threshold T_{SD}, create a new stratum and repeat the previous step.

Parameters T_{SD} and W_T allow to control the number of strata. There is a tradeoff between the number of strata and the gain in precision we can obtain from workload stratification. The degree of confidence obtained with workload stratification is shown in Figure 6 for a 4-core processor using the IPCT metric, $T_{SD} = 0.001$ and $W_T = 50$. It is very important to define strata separately and independently for each pair of microarchitectures and for each metric. For example, for 4 cores and WSU metric, the pair DRRIP-FIFO generates 34 strata, DRRIP-LRU generates 15 strata, and FIFO-RND generates 17 strata. We measured the degree of confidence experimentally by drawing 10000 different stratified samples for each sample size. For the pair FIFO-RND and a sample as small as 10 workloads, the degree of confidence with workload stratification is approximately 100 % while simple random sampling requires about 80 workloads to reach the same confidence. The pair DIP-LRU requires 50 workloads with workload stratification while random sampling requires 800 workloads to reach an equivalent confidence. The performance difference of DRRIP vs. FIFO is large enough for all sampling

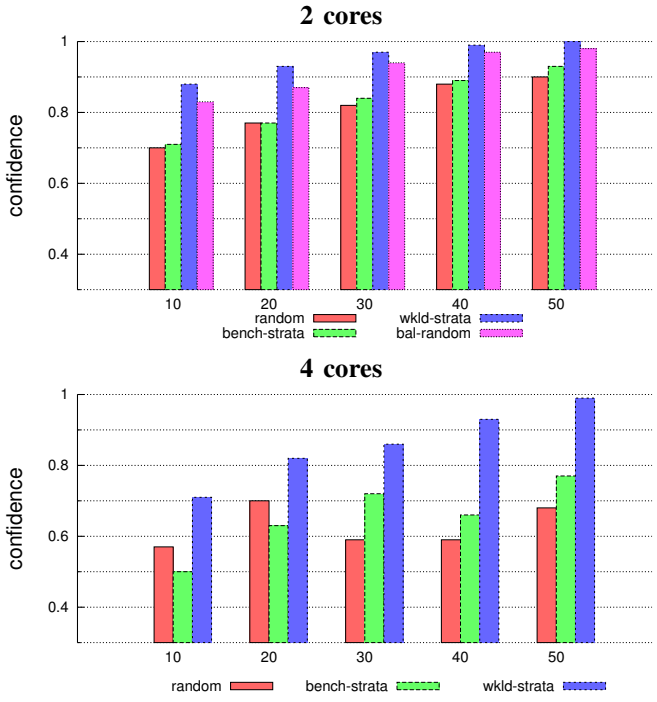

Fig. 7. Experimental degree of confidence measured with Zesto as a function of the sample size for $DIP > LRU$, 4 sampling methods (simple random sampling, balanced random sampling, benchmark stratification and workload stratification), using the IPCT metric.

methods to bring nearly 100 % of confidence with just 10 workloads.

C. Actual degree of confidence

The degrees of confidence presented in Figure 6 were estimated with BADCO in order to isolate the error coming from workload sampling from the error due to approximate simulation, i.e., as if BADCO were 100% accurate. However in practice the approximate simulator is also a source of inaccuracy. Figure 7 shows the experimental degree of confidence for DIP vs. LRU for small sample sizes and for the different sampling methods. Here the degree of confidence is measured with Zesto, but workload stratification is done with BADCO.

We did the experiment as follows. For 2 cores, we have simulated with Zesto the full population of 253 workloads. For 4 cores and 8 cores, we have simulated 250 workloads. For a given sample size and for each sampling method[6], we take 100 samples, each sample consisting of workloads that we have simulated with Zesto. We compute the per-sample throughput metric (here, the IPCT) for each of the 100 samples and for DIP and LRU. The experimental degree of confidence is the fraction of samples on which DIP outperforms LRU.

The results in Figure 7 confirm that the degree of confidence of samples selected with workload stratification outperform the degree of confidence of those selected with random, balanced random and benchmark stratification sampling methods. However, the degree of confidence measured with Zesto for workload stratification and 4 cores seems to be less than the degree of confidence estimated with BADCO for 4 cores on the pair LRU-DIP in Figure 6.

[6]We did not apply balanced random sampling for 4 cores and 8 cores because the method we used for automatically defining a balanced sample works with the full workload population. In real situations this would not be a problem because detailed simulations are normally done *after* the workload sample is defined.

VII. Summary and practical guideline

The method we propose relies on qualitatively accurate approximate simulation. It is not intended for design space exploration, but for studying incremental modifications of a microarchitecture, i.e., for comparing a baseline microarchitecture and a new microarchitecture. Moreover it is most useful when it is not obvious a priori whether the new microarchitecture outperforms the baseline. Detailed microarchitecture simulation is used to obtain information that the approximate simulator does not provide, such as power consumption (e.g., to find if the extra hardware complexity is worth the performance gain). In this situation, the two machines differ only in some parts of the microarchitecture that the approximate simulator should model precisely. The parts of the microarchitecture that are identical in both machines can be abstracted for simulation speed. For example, if one wants to compare two branch predictors for an SMT core, the approximate simulator should model the branch predictors precisely, but the other core mechanisms can be approximated.

Developing an ad-hoc approximate simulator requires some effort. Approximate simulators are commonly used in the industry for design space exploration, hence for some studies it may be sufficient to reuse and modify an already available approximate simulator. Publicly available approximate simulators include Sniper [8], recently developed at the University of Ghent, which can be used for various studies, e.g., uncore studies or branch prediction studies. If one wants to compare different uncore microarchitectures, an approximate simulation method such as BADCO is also possible. It took us roughly one person-month of work to implement the BADCO core models for this study.

Once we have a fast approximate simulator, we simulate a large workload sample for the two microarchitectures (balanced random sampling should be used so that all the benchmarks have the same weight, cf. Section VI-A). The required size for such sample does not depend on the full population size but on the actual coefficient of variation c_v (formula (8)). However, the actual c_v cannot be estimated with certainty from a workload sample. Nevertheless, the larger the sample, the more likely it is representative. For instance if $c_v < 10$, i.e., if the two microarchitectures are not equivalent throughput-wise, 800 random workloads are sufficient.

Then, assuming the large sample is representative of the full population, we estimate the coefficient of variation c_v on this sample. If c_v is greater than 10, we declare that the two machines offer the same average throughput. If c_v is less than 2, random sampling may be sufficient, as a few tens of workloads ensures a high confidence (cf. formula (8)). Nevertheless, for such small sample, balanced random sampling should be preferred over random sampling. It is when c_v is in the $[2, 10]$ range that we recommend using

workload stratification. However, one must keep in mind that the workload sample thus defined is valid only for a pair of microarchitectures and for a throughput metric.

A. Simulation overhead: example

As an example, let us consider the top graph of Figure 6 (DIP vs. LRU) and the speedup numbers provided in Table III, and let us assume that we simulate 100 million instructions per thread, i.e., 400 million instructions per workload. With balanced random sampling, 30 workloads yields a confidence of 75% and necessitate roughly $30 \times (400/0.049)/3600$ cpu*hours of Zesto simulation for each replacement policy, that is, 136 cpu*hours in total. To reach a confidence of 90% under balanced random sampling, we need 120 workloads, which requires $2 \times 120 \times (400/0.049)/3600 \approx 544$ cpu*hours. of Zesto simulation. That is, to increase the degree of confidence from 75% to 90%, we need 300% extra simulation. With workload stratification, 30 workloads are sufficient to obtain 99% of confidence, which takes 136 cpu*hours of Zesto simulation. In order to identify 30 "good" workloads, we first generate a BADCO model for each benchmark, which takes $22 \times 2 \times (100/0.17)/3600 = 7$ cpu*hours (22 benchmarks, 2 traces per benchmark, 100 million instructions, Zesto single-core simulation speed). Then we simulate 800 random workloads with BADCO for each policy (notice on this example that 600 random workloads are sufficient to reach 99% of confidence). This takes $2 \times 800 \times (400/1.89)/3600 = 94$ cpu*hours. Increasing the degree of confidence from 75% to 99% requires $(7+94)/136 \approx 74\%$ extra simulation with workload stratification. On this example, workload stratification yields more confidence than random sampling for a simulation overhead that is 4 times smaller.

VIII. Conclusion

The multiprogrammed workloads used in computer architecture studies are often defined without any clear method and with no guarantee that the chosen sample is representative of the workload population. Indeed, it is difficult to assess the representativeness of a workload sample without simulating a much larger number of workloads, which is precisely what we want to avoid by using sampling. We propose to solve this dilemma with approximate simulations that trade accuracy for simulation speed. Approximate simulation is generally used for design-space exploration. We have shown in this study that approximate simulation can also help selecting multiprogrammed workloads in situations requiring detailed microarchitecture simulations.

We have investigated several methods for defining multiprogrammed workloads. As a case study, we compared several multicore last-level cache replacement policies. We have shown that, unless we know a priori that the microarchitecture under study *significantly* outperforms (or underperforms) the baseline, it is not safe to simulate only a few tens of randomly chosen workloads. An approximate yet qualitatively accurate simulator, because it runs faster, allows to consider a much large number of workloads. We have proposed a method for defining, from a large workload sample, a smaller sample to simulate with a detailed simulator.

We have considered in this study the problem of defining workload samples that tell if a microarchitecture outperforms another, consistently with the full workload population. To our knowledge, the problem of defining workload samples that provide accurate speedups with high probability is still open.

References

[1] G. Loh, S. Subramaniam, and Y. Xie, "Zesto : a cycle-level simulator for highly detailed microarchitecture exploration," in *Proc. of the Int. Symp. on Performance Analysis of Systems and Software*, 2009.

[2] R. A. Velásquez, P. Michaud, and A. Seznec, "BADCO: behavioral application-dependent superscalar core model," in *12th Int. Conf. on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2012.

[3] K. V. Craeynest and L. Eeckhout, "The multi-program performance model : debunking current practice in multi-core simulation," in *Proc. of the IEEE Int. Symp. on Workload Characterization*, 2011.

[4] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for simultaneous multithreading processor," in *Proc. of the 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2000.

[5] K. Luo, J. Gummaraju, and M. Franklin, "Balancing throughput and fairness in SMT processors," in *Proc. of the IEEE Int. Symp. on Performance Analysis of Systems and Software*, 2001.

[6] H. Vandierendonck and A. Seznec, "Managing SMT resource usage through speculative instruction window weighting," *ACM Transactions on Architecture and Code Optimization*, vol. 8, no. 3, Oct. 2011.

[7] M. Van Biesbrouck, L. Eeckhout, and B. Calder, "Representative multiprogram workloads for multithreaded processor simulation," in *Proc. of the IEEE Int. Symp. on Workload Characterization*, 2007.

[8] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proc. of Supercomputing 2011*, 2011.

[9] S. Kanaujia, I. E. Papazian, J. Chamberlain, and J. Baxter, "FastMP : a multi-core simulation methodology," in *Workshop on Modeling, Benchmarking and Simulation*, 2006.

[10] J. Moses, R. Illikkal, R. Iyer, R. Huggahalli, and D. Newell, "ASPEN : towards effective simulation of threads & engines in evolving platforms," in *Proc. of the 12th IEEE/ACM Int. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, 2004.

[11] K. Lee, S. Evans, and S. Cho, "Accurately approximating superscalar processor performance from traces," in *Proc. of the Int. Symp. on Performance Analysis of Systems and Software*, 2009.

[12] F. Ryckbosch, S. Polfliet, and L. Eeckhout, "Fast, accurate, and validated full-system software simulation on x86 hardware," *IEEE Micro*, vol. 30, no. 6, pp. 46–56, Nov. 2010.

[13] L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell, "Exploring large-scale CMP architectures using ManySim," *IEEE Micro*, vol. 27, no. 4, pp. 21–33, Jul. 2007.

[14] P. Michaud, "Demystifying multicore throughput metrics," *IEEE Computer Architecture Letters*, Aug. 2012.

[15] W. G. Cochran, *Sampling Techniques, 3rd Edition*, 2nd ed. John Wiley, 1977.

[16] M. Qureshi, A. Jaleel, Y. Patt, S. C. Steely Jr., and J. Emer, "Adaptive insertion policies for high performance caching," in *Proc. of the 34th Int. Symp. on Computer Architecture*, 2007.

[17] A. Jaleel, K. Theobald, S. C. Steely Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *Proc. of the 37th Int. Symp. on Computer Architecture*, 2010.

[18] T. Austin, E. Larson, and D. Ernst, "SimpleScalar : an infrastructure for computer system modeling," *IEEE Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002, http://www.simplescalar.com.

[19] M. Van Biesbrouck, L. Eeckhout, and B. Calder, "Considering all starting points for simultaneous multithreading simulation," in *Proc. of the Int. Symp. on Performance Analysis of Systems and Software*, 2006.