Selecting Benchmarks Combinations for the Evaluation of Multicore Throughput

Ricardo A. Velasquez, Pierre Michaud, André Seznec

Selecting Benchmarks Combinations for the Evaluation of Multicore Throughput

Ricardo A. Velásquez, Pierre Michaud, André Seznec

RESEARCH REPORT
N° 737446
October 2012
Project-Teams ALF

ISSN 0249-6399   ISRN INRIA/RR--737446--FR+ENG

Selecting Benchmarks Combinations for the Evaluation of Multicore Throughput

Ricardo A. Velásquez, Pierre Michaud, André Seznec

Project-Teams ALF

Research Report n° 737446 — October 2012 — 23 pages

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 cycle-accurate microarchitecture simulators are slow, we want a set 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. We evaluate their effectiveness on a case study, the comparison of several multicore last-level cache replacement policies. We show that random sampling, the simplest method, is robust to define a representative sample of workloads, provided the sample is big 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.

Key-words: workload selection, multicore, sampling methods, replacement policies, throughput methods

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Sélection de charges multitâches de référence pour l'évaluation du débit d'exécution des processeurs multicœurs

Résumé : Aujourd'hui, la plupart des processeurs hautes performances sont capables d'exécuter plusieurs flots d'exécution simultanément. Ces flots d'exécution partagent les ressources du processeur, comme le cache de dernier niveau, ce qui peut réduire le débit d'exécution de manière difficilement prévisible, selon les caractéristiques de ces flots. Les architectes étudient généralement les charges multitâches en considérant un ensemble de charges de référence et des combinaisons de ces charges de référence. Comme les simulateurs précis au cycle près sont lents, nous voulons un ensemble de combinaisons qui soit aussi petit que possible, mais représentatif. Cependant, il n'existe pas de méthode standard pour la sélection de ces échantillons et différents auteurs ont utilisé différentes méthodes. Il n'est pas clair en quoi le choix d'un échantillon en particulier a une incidence sur les conclusions d'une étude. Nous proposons et comparons différentes méthodes d'échantillonnage permettant de définir des charges multitâches pour l'architecture des ordinateurs. Nous évaluons leur efficacité sur une étude de cas : la comparaison de plusieurs politiques de remplacement pour le cache de dernier niveau. Nous montrons que l'échantillonnage aléatoire, la méthode la plus simple, est robuste pour définir un échantillon représentatif de la charge de travail, à condition que l'échantillon soit assez grand. Nous proposons une méthode d'estimation de la taille de l'échantillon nécessaire basée sur une simulation rapide approximative. Nous proposons une nouvelle méthode, la stratification de charges multitâches, qui est très efficace pour réduire la taille de l'échantillon dans les cas où un échantillonnage aléatoire requerrait de grands échantillons.

Mots-clés : charges multitâches, multicœurs, méthodes d'échantillonnage, politique de remplacement

1 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 cycle-accurate 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, define a fixed set of multiprogrammed workloads from these benchmarks, simulate these workloads and use a throughput metric to quantify performance.

The population of possible benchmarks combinations may be very large. Hence most studies use a relatively small sample of a few tens, sometimes a few hundreds of workloads. Assuming that all the benchmarks are equally important, we would like this sample to be representative of the whole workloads population. 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 a workloads sample without simulating a much 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 approximate simulation can also help selecting representative multiprogrammed workloads for situations requiring the accuracy of cycle-accurate simulation.

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 very detailed microarchitecture simulator [9], and with BADCO, a fast approximate simulator [16].

We show that, unless we know a priori that one design point significantly outperforms the other, it is not safe to simulate only a few tens of random workloads, as currently done in many studies. Hence it is necessary to simulate a large sample of workloads, which is possible with a fast approximate simulator. We provide a method for determining, out of the large sample of workloads, what should be the size of a random set of workloads for cycle-accurate simulations. We propose an improved sampling method, balanced random sampling, which defines workloads in such a way that all the benchmarks occur the same number of times. Sometimes, random sampling necessitates more than a few tens of workloads. We have evaluated an alternative method, benchmark stratification, which defines workloads by first defining benchmarks classes. However, this method is not significantly better than random sampling. Finally, we propose a new method, called workload stratification, which is very effective at reducing the sample size when random sampling would require too big a sample.

To our knowledge, this study is the first to propose approximate simulation as a means for defining a set of multiprogrammed workloads to be used for studies requiring the accuracy of cycle-accurate simulation.

The paper is organized as follows. Section 2 presents the related work and the current practices. In Section 3, we propose a method for obtaining the size of a representative workloads sample under random sampling. Section 4 describes our experimental setup and briefly presents BADCO, our approximate simulator. We evaluate experimentally our random sampling method in Section 5. Then Section 6 introduces and evaluates experimentally three alternative sampling methods. Section 7 proposes a practical guideline. Finally, Section 8 concludes this study.

2 Second section

Simulation objectives for a computer architect are generally to compare two or more multicore designs under some criterion such as execution time, multiprogram throughput, power consumption, fairness, etc. Generally one wants also to quantify the differences between design points. 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 design points 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 cores are identical and assuming that the same benchmark can be replicated several times, there are $\binom{B+K-1}{K}$ possible workloads. Because accurate simulators are very slow, computer architects generally consider a *sample* of W workloads, where W is typically only a few tens. Hence most of the time, the fixed set of workloads is much smaller than the total number of possible workloads. Yet, there is no standard method for defining a representative set of workloads, although there seems to be some common practices. Nevertheless, the method used for selecting the W workloads may change the conclusions of a study dramatically.

The design points being compared are simulated on all W workloads. For each design point, we obtain a total of $W \times K$ IPC 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 design point that gives the highest throughput value on the W workloads is supposed to be the one offering the highest throughput on the full workloads population.

2.1 Current practices

We did a survey of papers published in 3 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 papers use a completely random selection of benchmarks. 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 author's 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 authors choose a single benchmark to be representative of its class.

2.2 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.

[1] Physical cores may be SMT

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

However, the behavior of modern superscalar processors is sometimes quite complex, and accurate 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 [6] that using only a few tens of random workloads, as seen in some studies, does not permit evaluating accurately a throughput metric like weighted speedup [14] or harmonic mean of speedups [10]. In their experiments, about 150 random 4-thread workloads are necessary to be able to compute throughput with reasonable accuracy out of 29 individual SPEC2006 benchmarks. That is, random selection requires a sample of workloads larger than what is used in most studies. That may be a reason why most authors 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 [15]. Van Biesbrouck et al. [3] described a fully automatic method to define workloads using microarchitecture-independent profiling data. Instead of classifying benchmarks, they apply cluster analysis directly on points representing workloads.

2.3 Approximate simulation

Several approximate microarchitecture simulation methods have been proposed [6, 4, 7, 11, 8, 16, 13, 17] (the list is not exhaustive). In general, these methods trade accuracy for simulation speed. They are usually advocated for design space exploration and, more generally, for situations where the slowness of cycle-accurate simulators limits their usefulness.

2.4 Throughput metrics

Several throughput metrics are commonly used in computer architecture studies. The most frequently used ones are the IPC throughput (IPCT), the weighted speedup (WSU) [14], and the harmonic mean of speedups (HSU) [10]. These throughput metrics can expressed with a single formula. The per-workload throughput 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 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 for the benchmark running alone on the reference machine (*single-thread IPC*).

3 Random sampling

As noted in Section 2, random sampling is not the most widely used 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 remaining of this section a model for estimating the probability of drawing correct conclusions under random workload selection, as a function of the sample size.

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 $N = \binom{B+K-1}{K}$ workloads. The problem of comparing two design points 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 for each design point. T_X and T_Y are two random variables. We define a new random variable D:

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

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

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

If we have some information about the distribution of D, we may be able to compute the probability that D is positive. When the A-mean is used in the throughput metric (IPCT or WSU), and because the W workloads are chosen randomly and independently from each other, the Central Limit Theorem applies, and D can be approximated by a normal distribution [5].

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 $\sigma^2 \times F/W$ with $F = \frac{N-W}{N}$ a finite population correction factor (sampling without replacement). The **degree of confidence** that Y is better than X is equal to the probability that D is positive:

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

where erf is the error function and $c_v = \sigma/\mu$ is the coefficient of variation of $d(w)$.

For the HSU metric, a H-mean is used, and it is the inverse of the HSU on which the Central Limit Theorem applies. Thus, we define variable D as

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

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

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

whose coefficient of variation c_v is used in the right-hand side of equation (5).

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

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

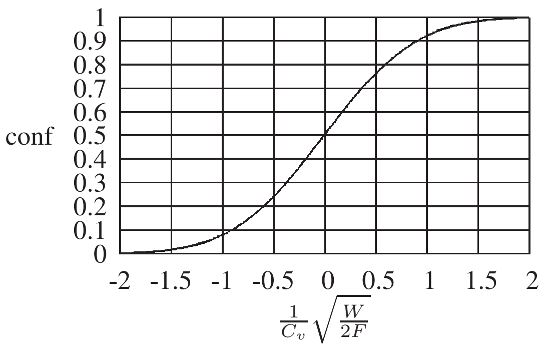


Figure 1: Confidence degree as a function of $\frac{1}{c_v}\sqrt{\frac{W}{2F}}$ (equation (5).

Solving this equation for W and assuming $N \gg W$ (i.e., $F \approx 1$) 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 measured from experiments. We present an experimental validation of the model in Section 5.1.

4 Experimental evaluation

4.1 Simulation setup

Two simulators are used in this work: Zesto [9], a cycle-accurate simulator, and BADCO [16], an approximate simulator that allows exploring a large number of workloads. The BADCO simulator is shortly presented in section 4.2.

Our experiments analyze the performance of multicore processors with 2, 4 and 8 identical cores. Table 1 presents a summary of cores characteristics. A case study with four uncore design points is evaluated, each design point corresponding to a different replacement policy in the shared last-level cache: LRU, RANDOM (RND), FIFO and DRRIP. Table 2 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 design point using BADCO for the full set of workloads whenever possible (253 workloads for 2 cores, 12650 workloads for 4 cores), or for a big sample when the number of possible combinations is huge (10000 workloads for 8 cores[3]). With Zesto, we also perform cycle accurate simulation for 250 randomly selected workloads for 2, 4 and 8 cores respectively and for every design point.

All the benchmarks were compiled with gcc-3.4 using the "-O3" optimization flag. For generating BADCO traces, we skip the first 40 billions instructions of each benchmark, and the trace represents the next 100 millions instructions (no cache warming is performed). We assume that simulations are reproducible, so that traces represent exactly the same sequence of dynamic μops. We used SimpleScalar EIO tracing feature [2], 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 millions instructions earlier than the other threads, it is restarted. This is done as many times as necessary until all the threads in the workloads have executed at least 100 millions instructions. Performance is measured only for the first 100 millions committed instructions of each thread.

4.2 BADCO

In [16], Velasquez et al. described a Behavioral Application-Dependent Superscalar Core Model (BADCO) for fast simulation of uncore configurations. As a behavioral core model, BADCO emulates the *external* behavior of the core (i.e. the way the core communicates with the uncore), not the mechanisms inside the core. Such behavioral model is derived from cycle-accurate 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

[3]Considering all the possible combinations of 22 benchmarks on 8 symmetric cores yields 4292145 workloads.

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 1: Core configuration.

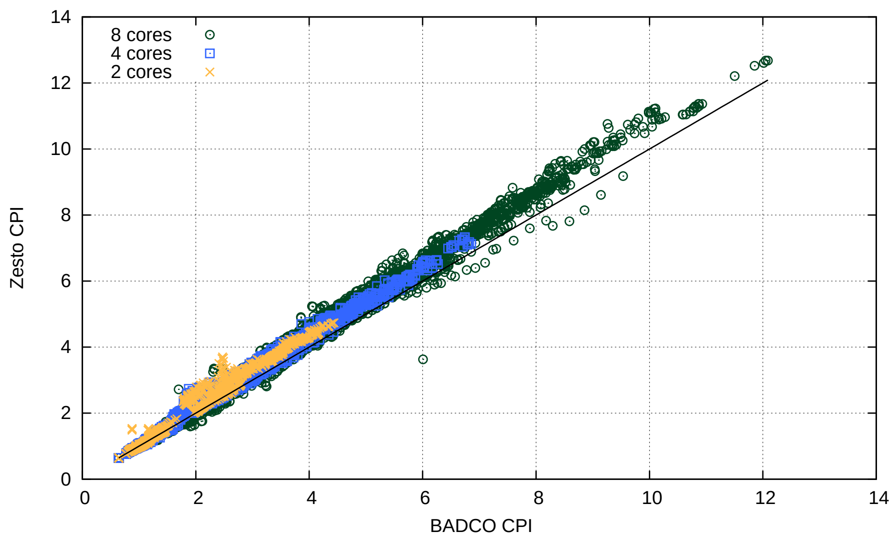


Figure 2: CPI measured with Zesto on the vertical axis versus estimated CPI with BADCO on the horizontal axis.

their associated uncore requests. The graph edges represent inferred dependencies 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*.

In [16], the authors have evaluated the BADCO method on single thread simulation. In the present work, we use BADCO models to simulate multiprogram workloads. Extending BADCO to execute multiprogram 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 multi-core running several independent threads simultaneously. We connect several BADCO machines, one per core, to a cycle accurate simulator of the uncore[4]. 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. Once this is done, the uncore processes the request. The uncore notifies the BADCO machine about the completion of one of its request through a call-back. A request

[4] The uncore simulator was extracted from Zesto.

	2 cores	**4 cores**	**8 cores**
LLC size/latency	1 MB / 5 cycles	2 MB / 6 cycles	4 MB / 7 cycles
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		

Table 2: Uncore configurations.

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

Table 3: BADCO average speedup for 1, 2, 4 and 8 cores.

completes when it has fully completed the processing through the cache hierarchy.

Figure 2 reports the measured and the estimated CPIs for Zesto and BADCO respectively. Each dot represent the CPI performance of a benchmark in one of 250 benchmark combinations. A perfect estimation would imply that all the dots lie on the bisector. In this case, we observe that most of the points are over the bisector. This indicates that BADCO tends to slightly underestimate the CPI. The average of the *absolute CPI error* is 4.62 %, 3.92 % and 4.05 % for 2, 4 and 8 cores respectively. The maximum error is in all cases less than 22 %. Moreover, 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 % 0.60 % and 0.94 % for 2, 4 and 8 cores respectively. BADCO is notably better in predicting speedups than raw CPIs.

Table 3 reports the *simulator* performance of Zesto and BADCO[5]. BADCO is clearly faster than the cycle accurate simulator Zesto, with simulation speedups going from 15x to 68x when going from 1 to 8 cores. It should be noted that BADCO still uses a cycle accurate uncore simulator.

[5] We do not count the time spent generating BADCO models.

5 Experimental results for random sampling

5.1 Random sampling model validation

We have validated experimentally the random sampling model described in Section 3 for all 6 pairs of replacement policies for all 3 metrics: IPCT, WSU and HSU; and for 2, 4, and 8 cores. The *experimental confidence degree* is defined as the probability that the sample throughput of the replacement policy Y to be greater than the sample throughput of replacement policy X. To compute this probability, we generate 10000 random samples for every sample size.

Due to the limited space, we just show the results of this experiment for one pair of policies and one metric. Figure 3 shows the degree of confidence that DRRIP outperforms LRU, as a function of the sample size W, and using the WSU metric. The model curve matches the experimental points quite well, even for small samples.

5.2 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 set of possible workloads with a reasonably high probability. As explained in Section 3, the coefficient of variation c_v of the random variable $d(w)$ (or $d'(w)$) 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. We show the value of $1/c_v$ measured with Zesto on a 250-workload sample and the value of $1/c_v$ measured with BADCO on the same 250-workload sample. The graph also shows the value of $1/c_v$ measured with BADCO on the full population of 12650 workloads. The sign of $1/c_v$ indicates which policy in the pair performs best. The magnitude $|1/c_v|$ gives an indication of how sharp the performance difference between the two policies.

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

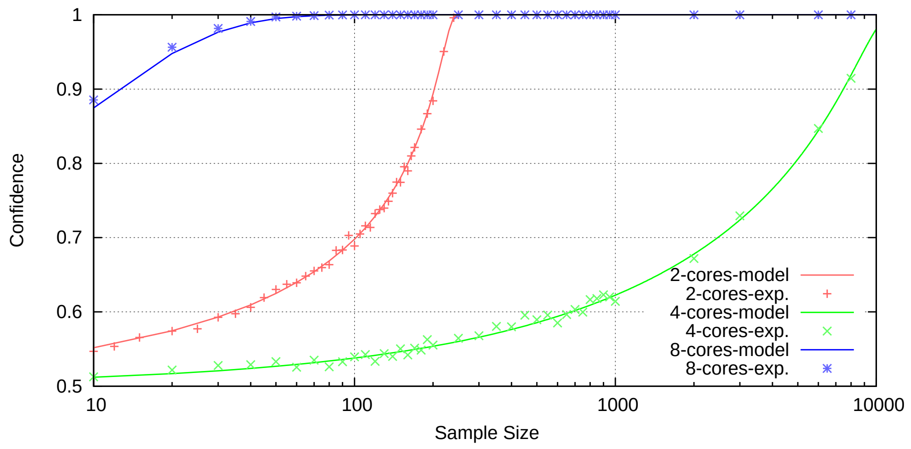
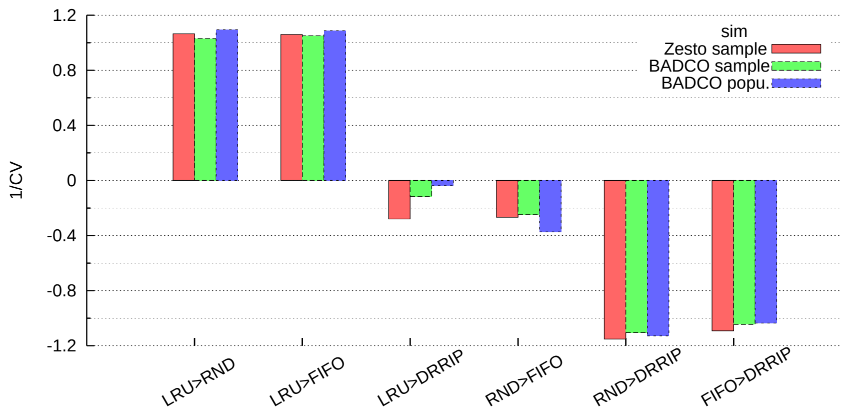
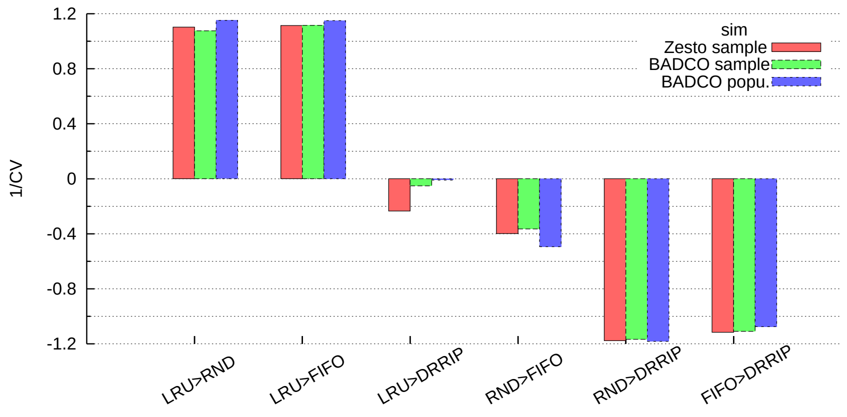


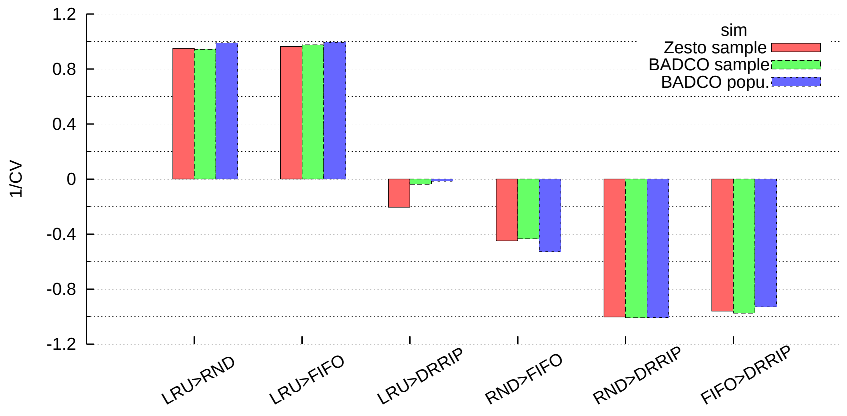
Figure 3: Confidence degree that *"DRRIP outperforms LRU"* as function of the sample size. Experimental result vs. analytical model. Throughput metric WSU.



(a) IPCT



(b) WSU



(c) HSU

Figure 4: Inverse of the coefficient of variation, $1/c_v$, assuming 4 cores processor. The 3 graphs corresponds to the 3 throughput metrics: IPCT, WSU and HSU. Each group of 3 bars corresponds to a pair of replacement policies being compared. The first bar gives $1/c_v$ measured with the 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 population of 12650 workloads.

However, when two policies have very close performance, such as LRU and DRRIP, $|1/c_v|$ 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. DRRIP is much smaller when computed on the full population than on the 250-workload sample. We cannot obtain a safe value of c_v on a small sample, unless we have the a priori knowledge that one design point significantly outperforms the other. Two design points may have the same performance on the full workload population, yet one design point may seem to outperform the other on a sample.

In other words, if we have no a priori reason to believe that one design point significantly outperforms the other, we must consider a sample of workloads as large as possible. A fast approximate simulator which is qualitatively accurate, such as BADCO, allows to consider a very large sample of workloads. On this large sample, we compute $|1/c_v|$. If $|1/c_v|$ is much smaller than 1 (say $c_v > 10$), we conclude that the two policies perform equally. Otherwise, we determine the sample size that is sufficient and we take a random sample of this size that we simulate with a cycle-accurate simulator in order to obtain a speedup value.

5.3 Different metrics may require different sample sizes

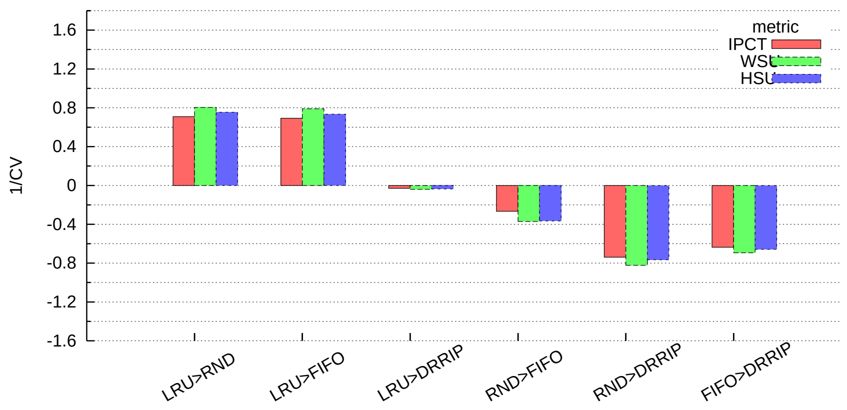
The physical meaning of a throughput metric depends on some assumptions regarding the benchmarks and what they represent. Different metrics rely on different assumptions. Some computer architecture studies use several throughput metrics to show the robustness of their conclusion. Yet, it is possible in theory that using a different metric may yield a different conclusion.

Figure 5 shows the inverse of the coefficient of variation, $1/c_v$, for different pairs of policies and for the throughput metrics: IPCT, WSU and HSU. *In these experiments*, the sign of c_v does not depend on the throughput metric. That is, assuming we use a large enough number of workloads, all 3 metrics rank the policies the same way. However, the magnitude of $|c_v|$ is not the same for all metrics. It means that some metrics permit using fewer workloads.

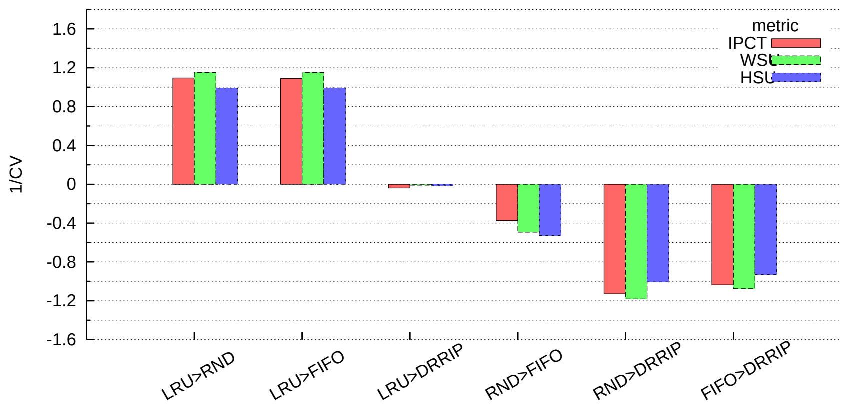
For example, when comparing RND and FIFO on 2, 4 and 8 cores, $|1/c_v|$ is greater with the WSU and HSU metrics than with the IPCT metric. For example, with 4 cores, IPCT requires a sample of $W \approx 57$ workloads, while WSU and HSU requires samples of $W \approx 33$ and $W \approx 29$ workloads respectively. The most effective metric for discriminating the two policies is HSU.

Nevertheless, it is difficult to find a priori which metric will be the most effective. We did not find any simple rule for predicting how different metrics will behave. For instance, HSU is the most effective metric for assessing that FIFO outperforms RND for 4 and 8 cores, but it is the least effective metric for assessing that DRRIP outperforms RND. This is not quite intuitive. A careful analysis of the reasons why one replacement policy outperforms another would probably help understand why a metric is more effective than another. But intuition may be misleading.

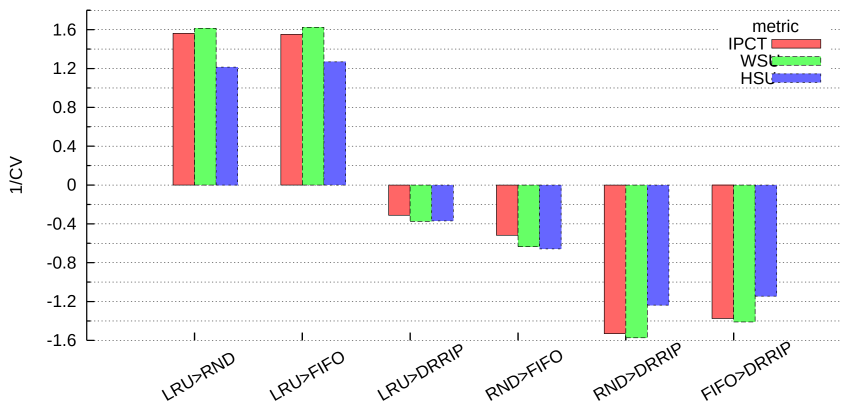
Moreover, we may not necessarily want to use the most effective metric in a study, as different metrics have different meanings. One may prefer to set the sample size according to the least effective metric, so that the sample is large enough for several different metrics.



(a) 2 cores



(b) 4 cores



(c) 8 cores

Figure 5: Inverse of the coefficient of variation, $1/c_v$, measured with BADCO on the full population of 12650 workloads. The 3 graphs corresponds to 2, 4 and 8 cores configurations. Each group of 3 bars corresponds to a pair of replacement policies being compared. The first bar gives $1/c_v$ measured for metric IPCT. The second bar gives $1/c_v$ measured for metric WSU. The third bar gives $1/c_v$ measured for metric HSU.

6 Alternative Sampling Methods

6.1 Balanced Random Sampling

If we consider the full population of workloads and count how many times a given benchmark occurs in these workloads, 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 can be selected multiple times (though this is unlikely). 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, the remaining 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 confidence degree. Figure 6 shows the confidence degree for several different sampling methods, including random sampling and balanced random sampling (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 Figure 6b where the required sample size is very large.

6.2 Stratified Random Sampling

Simple random sampling, as considered in Section 3, selects benchmarks with uniform probability in the workloads population. However, the workload population is generally not homogeneous. For example, let us assume that, on a subset of 80% of workloads, design point Y outperforms X, while on the 20% other workloads it is X that outperforms Y. 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* [5]. The method generalizes as follows.

With stratified sampling, 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

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

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:

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

where WX-mean stands for WA-mean or WH-mean and where the weight for stratum S_h is N_h/N.

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

Table 4: Classification of SPEC benchmarks according to memory intensity: Low $(MPKI < 1)$, Medium $(MPKI < 5)$, and High $(MPKI \geq 5)$.

If the strata are well defined, it may be possible to divide a very heterogeneous set of workloads into strata that are internally homogeneous, so that the coefficient of variation c_{v_h} 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.

Many different methods for constructing strata can be devised. Ideally, we would like to have the minimum number of strata that produce maximum precision and minimum W_h. The methodology must also establish the proportion $\frac{N_h}{N}$ of each stratum with respect to the full population. It is important to note that stratified sampling requires to drawn samples for each stratum. Hence the minimum value for W is equal to the number of strata. In the remaining of section, we compare two different methods for defining strata: benchmark stratification and workload stratification.

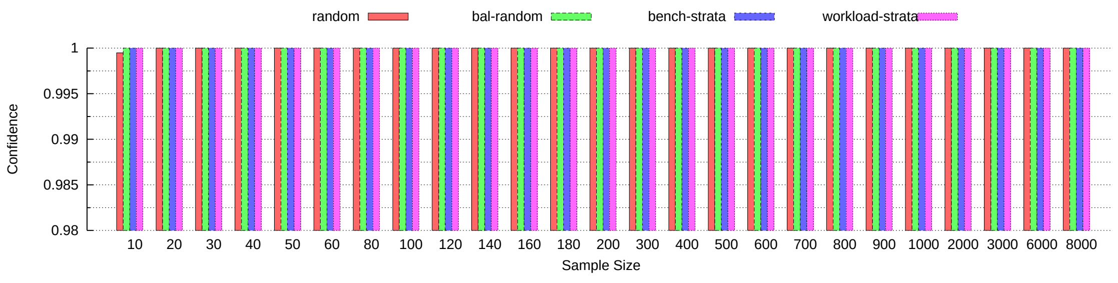
6.2.1 Benchmark Stratification

It is a common practice that computer architects define benchmark classes according to certain criteria. A common method is classifying the benchmarks as memory bound, CPU bound or cache sensitive [1]. Another method, used in cache studies, is to classify benchmarks according to the memory-intensity in misses per kilo-instruction (MPKI) [12]. The main assumption is that benchmarks in the same class have similar performance when sharing resources. Benchmark classes by themselves do not constitute strata. However, the classes permit constructing workloads strata mechanically. 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 the same class constitute a stratum. We can represent a stratum with an n-tuple $(c_1, c_2, ..., c_M)$, where the $c_1, c_2, ..., c_M$ are the number of occurrences of classes $C_1, C_2, ..., C_M$ in a workload, M is the number of classes and $\sum_{i=1}^{M} c_i = K$, the number of cores. Workloads with the same number of occurrences per class belong to the same stratum. The total number of strata L is equal to $\binom{M+K-1}{K}$. The size of the stratum is given by

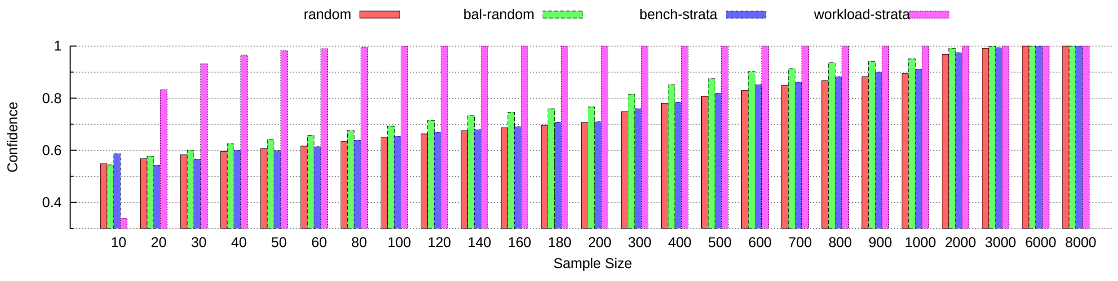
$$N_h = \prod_{i=1}^{C} \binom{b_i + c_i - 1}{c_i}$$

where b_i is the number of benchmarks in class C_i.

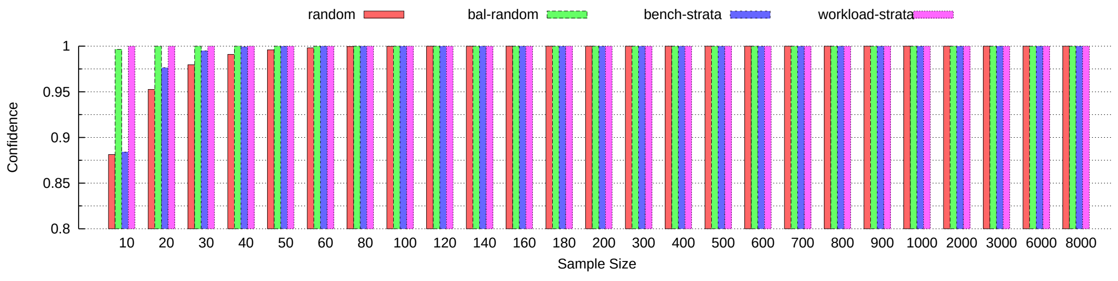
Table 4 shows a typical classification of the SPEC CPU2006 benchmarks according to the memory intensity measured in MPKI. For a 4 core processor, this classification generates 15 strata, hence $(c_{low} c_{med} c_{high})$ = (004, 013, 022, 031, 040, 103, 112, 121, 130, 202, 211, 220, 301, 400). Using this stratification setup, we want to compute the degree of confidence that $DRRIP > LRU$, $DIP > LRU$ and $DRRIP > DIP$. We have drawn 10000 stratified samples and have computed experimentally the confidence degree. In Figure 6, we compare the confidence degree of random sampling against stratified sampling with benchmark stratification, as a function of the sample size W. The results show, that for almost all sample sizes, stratified sampling with benchmark stratification increases the confidence degree to some extent, but does not reduce dramatically the sample size required to reach a high degree of confidence.



(a) $DRRIP > FIFO$



(b) $DRRIP > LRU$



(c) $FIFO > RND$

Figure 6: Alternative sampling methods. Confidence degree as function of the sample size for (a) $DRRIP > FIFO$, (b) $DRRIP > LRU$, and (c) $FIFO > RND$. Throughput metric $IPCT$ for 4 core configuration. Each group of 4 bars corresponds to samples size. The first bar gives the model computed confidence for simple random sampling. The second bar gives the experimental confidence for balanced random sampling. The third bar gives the experimental confidence for benchmark stratification. The fourth bar present the experimental confidence for workload stratification

It should be noted that the benchmark stratification method described here is an attempt to formalize some common practices which are diverse and not always rigorous. The studies we are aware of that define multiprogram workloads by first defining benchmarks classes do not use stratified sampling and formula (9).

6.2.2 Workload Stratification

Approximate simulators such as BADCO allow to estimate the throughput either for the full workload population, or for large samples of thousands of workloads. Once the throughput has been computed for all workloads, defining strata directly from workloads throughput values is straightforward.

There is not a single way to define the strata from throughput values. As we are interested in comparing two design points, we have used the distribution of $d(w)$ (see section 3) to create strata. The method we implemented is as follows:

1. Select two design points to be compared and compute $d(w)$ (or $d'(w)$ if the throughput metric uses a H-mean).
2. Sort $d(w)$ in ascending order.
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 permit controlling the number of strata. There is a tradeoff between the number of strata and the gain in precision we can obtain from workload stratification.

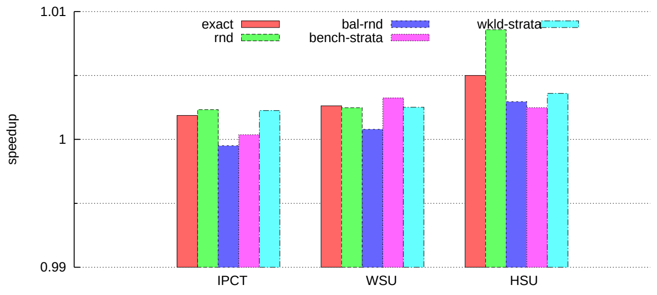
In Figure 6, we compare the confidence degree for workload stratification with the other sampling methods (random sampling, balanced random sampling, benchmark stratification), as a function of the sample size W. The results were obtained for a 4 core processor using the IPCT metric, $T_{SD} = 0.001$ and $W_T = 50$. It is very important to define strata separately and independently for each pair of design points being compared. The pair DRRIP-FIFO generates 34 strata, DRRIP-LRU generates 18 strata, and FIFO-RND generates 17 strata. We measure the confidence degree experimentally by drawing 10000 different stratified samples for each sample size. Results show that for the pair FIFO-RND (Figure 6c) and a sample as small as 10 workloads, the confidence degree for stratified sampling is approximately 100 %. In contrast simple random sampling requires around 80 workloads to reach the same confidence. The pair DRRIP-LRU (Figure 6b) with workload stratification requires a sample around 100 workloads, while the random sampling require around 6000 to reach an equivalent confidence. The difference in performance between DRRIP-FIFO (Figure 6a) is big enough for all the sampling alternatives to reach a confidence of 100 % with just 10 workloads.

6.3 Speedups computed with Zesto

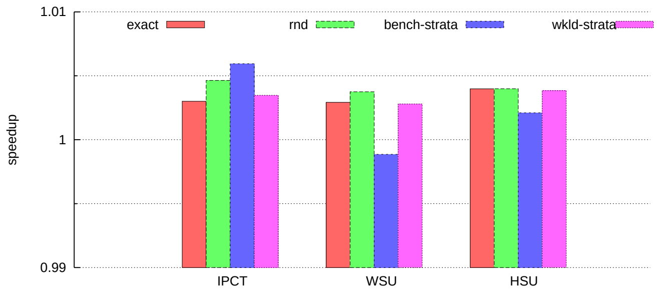
Figure 7 show the speedup of DRRIP vs. LRU obtained with Zesto for 2 cores and 4 cores. For 2 cores, we were able to simulate the full 253 workloads population and we have the exact speedup ("exact"). For 4 cores, we call "exact" speedup the speedup measured on the 250 workloads we have simulated with Zesto. The 20-workloads samples obtained with the different sample selection methods (“rnd”, “bal-rnd”, “bench-strata” and “wkld-strata”) are a subset of these 250 workloads[6].

As can be seen, for 20 workloads samples, there is high variability of speedups with respect to the reference bar identified as “exact”. In some cases, the samples even report slowdowns. This is the case of balanced random sampling in Figure 7a and IPCT metric. The same occurs for benchmark stratification in Figure 7b and WSU metric. Workload stratification is the sample selection method which provides the samples with closest speedups to the reference.

[6] We could not apply balanced random sampling for 4 cores because the method we implemented for automatically defining a balanced sample works with the full population. In practical utilizations of balanced random sampling, this would not be a problem because cycle-accurate simulations are normally done *after* the workload has been defined.



(a) 2 cores



(b) 4 cores

Figure 7: Speedups of DRRIP vs. LRU obtained with Zesto for 2 (a) and 4 (b) cores; and for throughput metrics IPCT, WSU and HSU. The bars represent different workload sample selection methods.

7 Summary and practical guideline

The method we propose relies on 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. Cycle-accurate simulation is used to obtain the speedup of the new microarchitecture over the baseline (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 as accurately as possible. 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 as accurately as possible, 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 [4], recently developed at the University of Ghent, which can be used for various studies, e.g., uncore studies, branch prediction studies, etc. If one wants to compare different uncore microarchitectures, an approximate simulation method such as BADCO is also possible. It took us roughly 1 person-month of work to implement the BADCO core models for this study.

Once we have the approximate simulator, we simulate a large sample of workloads for the two microarchitectures. If possible, benchmarks in this large sample should be balanced (cf. Section 6.1). A typical value for such large sample is 1000 workloads, i.e., at least an order of magnitude larger than what we can simulate with the cycle-accurate simulator.

Then we compute the coefficient of variation c_v on this large sample, as described in Section 3. If c_v is greater than 10, we declare the two microarchitectures equivalent performance wise. 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 thus defined is valid only for that pair of microarchitectures. If we want to evaluate a third microarchitecture, we must compare it with the baseline as explained above. We do not recommend using benchmark stratification.

8 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 whole workload population. Because of the slowness of cycle-accurate simulators, many authors use small workloads samples containing only a few tens of workloads.

It is difficult to assess the representativeness of a workloads sample without simulating a much larger number of workloads, which is precisely what we want to avoid.

We propose to solve this problem by using approximate simulation methods 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 the accuracy of cycle-accurate simulation.

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 one knows a priori that one design point significantly outperforms the other, it is not safe to simulate only a few tens of random workloads. We have shown that the frequent practice of defining workloads by first categorizing benchmarks into classes is not safe.

An approximate simulator, because it runs faster, permits considering a much large number of workloads than cycle-accurate simulation. We have proposed a method for defining, out of this large sample, some workloads to be simulated with the cycle-accurate simulator. If the performance of the two design points appears to be significantly different on the large sample, then it is sufficient to simulate a relatively small random set of workloads with the cycle-accurate simulator. We have proposed balanced random sampling, a method more robust than random sampling. Balanced random sampling defines workloads such that all the benchmarks occur the same number of times. We have introduced workload stratification, a new method for selecting workloads which is useful when random sampling would require more than a few tens of workloads. Workload stratification can be implemented in a quasi-automatic fashion.

Contents

References

[1] Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proc. of the 41st annual IEEE/ACM International Symposium on Microarchitecture*, 2008.

[2] T. Austin, E. Larson, and D. Ernst. SimpleScalar : an infrastructure for computer system modeling. *IEEE Computer*, 35(2):59–67, Feb. 2002. http://www.simplescalar.com.

[3] M. V. Biesbrouck, L. Eeckhout, and B. Calder. Representative multiprogram workloads for multithreaded processor simulation. In *Proc. of the IEEE International Symposium on Workload Characterization*, 2007.

[4] 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.

[5] W. G. Cochran. *Sampling Techniques, 3rd Edition*. John Wiley, 2 edition, 1977.

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

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

[8] 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.

[9] 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.

[10] K. Luo, J. Gummaraju, and M. Franklin. Balancing throughput and fairness in SMT processors. In *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2001.

[11] 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.

[12] S. P. Muralidhara, L. Subramanian, O. Mutlu, M. Kandemir, and T. Moscibroda. Reducing memory interference in multicore systems via application-aware memory channel partitioning. In *Proc. of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, 2011.

[13] F. Ryckbosch, S. Polfliet, and L. Eeckhout. Fast, accurate, and validated full-system software simulation on x86 hardware. *IEEE Micro*, 30(6):46–56, Nov. 2010.

[14] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for simultaneous multithreading processor. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000.

[15] H. Vandierendonck and A. Seznec. Managing smt resource usage through speculative instruction window weighting. *ACM Transactions on Architecture and Code Optimization*, 8(3), Oct. 2011.

[16] R. A. Velásquez, P. Michaud, and A. Seznec. BADCO: behavioral application-dependent superscalar core model. In *To appear in the 12th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2012.

[17] L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell. Exploring large-scale CMP architectures using ManySim. *IEEE Micro*, 27(4):21–33, July 2007.

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399