# Constant-work multiprogram throughput metrics for microarchitecture studies

Pierre Michaud


# Constant-work multiprogram throughput metrics for microarchitecture studies

**Pierre Michaud**

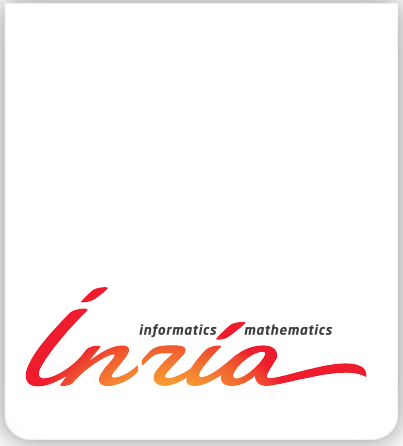
**RESEARCH REPORT**

**N° 8150**

November 2012

Project-Team ALF

ISSN 0249-6399 ISRN INRIA/RR--8150--FR+ENG




# Constant-work multiprogram throughput metrics for microarchitecture studies

Pierre Michaud

Project-Team ALF

Research Report n° 8150 — November 2012 — 21 pages

**Abstract:**
Multicore processors can improve performance by decreasing the execution latency of parallel programs, or by increasing throughput, i.e., the quantity of work done per unit of time when executing independent tasks. Throughput is not necessarily proportional to the number of cores and can be impacted significantly by resource sharing in several parts of the microarchitecture. Quantifying the impact of resource sharing on throughput requires a throughput metric. A majority of microarchitecture studies use equal-time throughput metrics, such as IPC throughput or weighted speedup, that are based on the implicit assumption that all the jobs execute for a fixed and equal time. We argue that this assumption is not realistic. We propose and characterize some new throughput metrics based on the assumption that jobs execute a fixed and equal quantity of work. We show that using such equal-work throughput metric may change the conclusion of a microarchitecture study.

**Key-words:** Microarchitecture studies, multicore processor, throughput metric

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

# Métriques de débit à travail constant pour les études en microarchitecture

**Résumé :** Les processeurs multi-coeurs augmentent les performances en réduisant le temps d'exécution des programmes parallèle ou en augmentant le débit, c'est-à-dire la quantité de travail effectuée par unité de temps lorsqu'on exécute des tâches indépendantes. Le débit n'est pas toujours proportionnel au nombre de coeurs, il peut dépendre fortement des conflits sur les ressources partagées de la microarchitecture. Les métriques de débit permettent au microarchitecte de quantifier l'impact de ces conflits sur le débit. Une majorité d'études en microarchitecture utilisent des métriques de débit telles que le débit d'instructions ou la somme des accélérations qui sont basées sur l'hypothèse implicite que toutes les tâches s'exécutent pendant des temps constants et égaux. Nous pensons que cette hypothèse n'est pas réaliste. Nous proposons et caractérisons de nouvelles métriques de débit basées sur l'hypothèse que les tâches exécutent des quantités de travail constantes et égales. Nous montrons que l'utilisation de ces nouvelles métriques de débit peuvent changer les conclusions de certaines études.

**Mots-clés :** Études en microarchitecture, processeur multi-coeur, métrique de débit

# 1 Introduction

Multicore processors can improve performance in two ways: they can decrease the execution latency of parallel programs, or they can increase throughput, i.e., the quantity of work done per unit of time when executing multiprogram workloads made of independent threads. While the importance of parallel programs in the current or future software ecosystem may be discussed, the benefit of higher throughput is undisputable and has been mostly visible in data centers and in the server side of the internet. However, threads running simultaneously on a multicore share some resources like the last-level cache, pin bandwidth, chip thermal design power, etc. With simultaneous multithreading (SMT), threads share core microarchitecture structures: level-1 caches, branch predictors, physical registers, scheduling logic, execution units, etc. Because of resource sharing, throughput is not necessarily proportional to the number of threads running simultaneously. Moore's law is expected to continue in the near future, and with it the growing of the number of cores on a single chip, leading to so-called "manycore" chips. Commercial manycore chips existing so far exploit data parallelism and have found an important niche in graphics processing, i.e., mostly on the client side of the internet. What is not clear is whether chips with several hundreds of general-purpose cores will be useful on the server side. Indeed, some of the shared resources are unlikely to grow linearly with the number of cores, especially pin bandwidth. Microarchitects studying general-purpose multicores or manycores are interested in quantifying throughput, to understand, and possibly decrease, the performance loss due to resource sharing.

We argue in this paper that the multiprogram throughput metrics that are commonly used for microarchitecture studies are deficient. We propose new throughput metrics aimed at correcting these deficiencies.

This paper is organized as follows. In Section 2 we list the most frequently used multiprogram throughput metrics and we explain why, in our opinion, they are deficient. We introduce in Section 3 a new throughput metric, the EW-IPC, which is based on the assumption of equal-work jobs. We show that the EW-IPC may lead to throughput paradoxes: accelerating a thread might sometimes decrease throughput. Because there is no simple analytical formula for the EW-IPC, we introduce in Section 4 some practical proxies for the EW-IPC: the H-IPC, ASH-IPC and SSH-IPC. We discuss in Section 5 the possible implications of using equal-work throughput metrics and we show with an example that the conclusions of a study may change dramatically. We conclude this study by recommending to use the ASH-IPC or the SSH-IPC in microarchitecture studies concerned with multiprogram throughput.

# 2 Multiprogram throughput metrics are broken

Multiprogram throughput is generally measured by considering a set of single-thread benchmarks and running simultaneously some combinations of benchmarks, called *workloads* in the rest of this study. In general, if the processor can run up to $K$ threads simultaneously, each workload is a combination of $K$ benchmarks (not necessarily distinct). It is customary in microarchitecture studies to define a fixed set of workloads and to simulate each workload separately *for a fixed time interval*. From simulations, a per-thread performance number is obtained for each workload, and a global throughput number is obtained by aggregating the per-workload performance numbers.

Obtaining a global throughput number is important because this is sometimes the only way to decide whether a mechanism should be implemented or not. Indeed, multiprogram throughput metrics are often used to study arbitration mechanisms in microarchitectures where one or several resources are shared between threads running simultaneously. Different arbitration mechanisms lead to different ways to allot finite resources to competing threads. Frequently, this leads to

modify the arbitration mechanisms in such a way that certain threads are accelerated at the cost of slowing down some other threads. Looking at individual threads or individual workloads may yield some insight when doing a study, but eventually it is global throughput that permits deciding which arbitration mechanism should be implemented.

So far, the three most frequently used multiprogram throughput metrics in microarchitecture studies are the so-called *IPC throughput*, *weighted speedup* and *harmonic mean of speedups*. These three metrics differ in how they define the per-workload performance.

## 2.1 The ET-IPC (aka IPC throughput)

IPC throughput defines the performance of a given workload as the sum of the IPCs (instructions executed per cycle) of each thread in the workload. Then, global throughput is obtained by averaging the per-workload performances:

$$\text{ET-IPC} = \frac{1}{W}\sum_{w=1}^{W}\sum_{i=1}^{K} IPC(w,i) \tag{1}$$

where $K$ is the number of logical cores, $W$ is the number of workloads and $IPC(w,i)$ is the IPC of the thread running on core $i$ in the $w^{th}$ workload. In the rest of this study, we denote the IPC throughput the *equal-time IPC* (ET-IPC), as each workload is run for a fixed and equal time interval. The ET-IPC was first used in the 1990's for studying SMT microarchitectures [13, 12], and is still one of the three most popular throughput metrics. However, blindly maximizing the ET-IPC may lead to implement resource arbitration mechanisms that favor high-IPC threads, penalizing low-IPC ones [12]. This led to the definition of alternative throughput metrics, in particular the *weighted speedup* [10, 7, 9, 11].

## 2.2 The weighted speedup and harmonic mean of speedups are inconsistent

Instead of raw IPCs or execution times, researchers often prefer to consider *speedups*, that is, relative performance [5]. Weighted speedup defines per-workload performance as the sum of the speedups of each thread constituting the workload, that is, the performance of workload $w$ is:

$$\sum_{i=1}^{K}\frac{IPC(w,i)}{IPC_{ref}(w,i)}$$

where $IPC_{ref}(w,i)$ is a reference IPC for the thread running on core $i$ in the $w^{th}$ workload (for instance, $IPC_{ref}(w,i)$ may be defined as the IPC of the thread when it runs alone on a reference machine).

Weighted speedup is one of the three most frequently used multiprogram throughput metrics. However, we have shown in a recent study that weighted speedup is *inconsistent*, as it gives more weight to benchmarks with a low reference IPC [6]. One may want to weight different benchmarks differently, but this should be stated explicitly and this must be justified. To our knowledge, none of the past studies that have used weighted speedup have provided a reason for weighting different benchmarks differently, perhaps because they were not aware of the inconsistency problem. The *harmonic mean of speedups*, the third most frequently used throughput metric, is inconsistent as well [6].

Using an inconsistent metric may lead to artificial conclusions, as illustrated in Table 1. The inconsistency of the weighted speedup and harmonic mean of speedups stems from using

| | **single thread** | **machine X running AB** | **machine Y running AB** |
|---|---|---|---|
| benchmark A | $IPC_{ref} = 0.2$ | $IPC = 0.1$ | $IPC = 0.2$ |
| benchmark B | $IPC_{ref} = 1$ | $IPC = 0.2$ | $IPC = 0.1$ |
| weighted speedup | | 0.7 | 1.1 |
| harmonic mean of speedups | | 0.29 | 0.18 |

Table 1: Example illustrating the inconsistency of weighted speedup and harmonic mean of speedups. Benchmarks A and B are equally important. Without further information on benchmarks, the only possible conclusion is that machines X and Y offer the same throughput.

a unit of work which is not the same for all the benchmarks and which depends on a reference machine, the machine on which reference IPCs are measured. Troughput is defined as the quantity of work done per unit of time. A sound throughput metric, one which can be used to compare machines, should define the unit of work independently from a reference machine, as the choice of a particular reference machine is arbitrary (the choice of a different reference machine could change the conclusions). If one wants, for whatever reason, to give more weight to some benchmarks, such weighting should be done consciously and should be justified.

## 2.3 What about geometric mean of speedups ?

It is well known that, in the case of single-thread performance, speedups should be aggregated with a geometric mean in order to avoid consistency problems [2, 5], and researchers generally use the geometric mean to summarize single-thread speedups. Oddly, in the case of multiprogram throughput, the practice of adding speedups (as in weighted speedup) or taking their harmonic mean is still widespread.

Though we do not exclude completely the possibility to define a *meaningful* throughput metric using a geometric mean, it is not clear how such metric should be defined. The geometric mean of speedups gives an estimate of the median speedup of random programs under the assumption that benchmarks are representative and that speedups are distributed log-normally [5]. That is, an assumption is made not only on programs (as in all performance metrics) but also on machines. It is easy to imagine cases where this assumption does not hold[1]. The assumption of log-normality cannot be true in general but coincidentally [4].

## 2.4 The SPECrate throughput metric

SPEC defines a throughput metric called SPECrate, based on running homogeneous workloads, i.e., running several independent copies of the same benchmark and measuring how long it takes for all copies to finish executing [1]. However, SPECrate is limited to homogeneous workloads. It is not an appropriate metric for studying microarchitectures that exploit the possible heterogeneity of behavior among concurrently running threads. For example, if we use only homogeneous workloads to evaluate multiprogram throughput, we may be led to conclude that a shared last-level cache offers little throughput advantage over private caches, overlooking the possibility for a shared cache to exploit the fact that different applications may have different cache space requirements [8].

[1] For example, consider a processor X such that IPCs are distributed log-normally, with a median IPC equal to 1. That is, 50% of programs have an IPC greater than 1. Then consider a processor Y that can issue only a single instruction per cycle, so that 50% of programs have an IPC close to 1. The IPC distribution of processor Y is not log-normal, neither the speedups.

### 2.5 Equal-time throughput metrics

A meaningful throughput metric is associated with a *throughput experiment* such that the quantity of work per unit of time measured with this experiment is equal to the throughput value given by the metric.

Both the ET-IPC and the weighted speedup metrics are *equal-time* throughput metrics: the throughput experiment with which they are associated divides the execution of a set of independent jobs into *equal time* intervals such that, during a time interval, a single workload is running continuously (that is, no context switch happens)[2].

Hence if a set of workloads is defined independently from any particular machine, and if all the benchmarks contribute equally to the jobs constituting the workloads, all the benchmarks run for the same total time, *whatever the machine's performance*.

There exists some applications that actually behave like that, i.e., they try to do as much work as possible during a fixed time. This typically corresponds to some interactive or real-time applications that can adapt the quality of their output as a function of the machine's performance. Such applications produce jobs that may be termed *constant-time* jobs. However, this concerns a minority of applications.

Most jobs, including batch jobs but also interactive and real-time jobs, are *constant-work* jobs. A constant-work job has fixed work to do, which typically corresponds to a fixed number of program instructions to execute, and the time to do this work depends on the machine's performance.

In our opinion, the implicit assumption of constant-time jobs in the conventional "IPC throughput" metric is the main drawback of that metric. Replacing raw IPCs with speedups, as in weighted speedup, not only makes the metric inconsistent but does not solve the basic problem: the quantity of work done by a job still depends on the machines performance.

To solve this issue, we propose in the rest of this study some new multiprogram throughput metrics that are based on the assumption that jobs execute a fixed and equal quantity of work. Our goal is to state the assumptions clearly and to characterize the metrics we propose, unlike what has been done with other throughput metrics.

## 3 The EW-IPC: an equal-work IPC throughput metric

We propose to define an equal-work throughput metric for microarchitecture studies based on the following throughput experiment:

> (1) The unit of work is the instruction
> (2) The behavior of a job is similar to one of the benchmarks, chosen randomly, and all the benchmarks are equally likely
> (3) All the jobs execute a fixed and equal number of instructions
> (4) There is a single job queue, which is never empty
> (5) The scheduling policy is first-in first-out (FIFO, aka first-come first-served)

We assume that the unit of work is the instruction, which is a natural unit of work for the microarchitect. This brings consistency: all the benchmarks are treated equally. Hence throughput is given in instructions per unit of time, e.g., instructions per cycle (for a fixed clock cycle) or

[2]The only difference between ET-IPC and weighted speedup is that they use different units of work. The ET-IPC assumes that the unit of work is the instruction, and that one instruction from one benchmark is worth one instruction of another benchmark. The weighted speedup assumes that the unit of work for a benchmark is the average number of instructions executed in a clock cycle by the benchmark when it runs alone on a reference machine (hence different units of work for different benchmarks).

instructions per second. In the rest of this study, we consider throughput in instructions per cycle.

The second assumption makes sense because, unless we have some specific information about the representativeness of a benchmark (which is rarely the case), we have no reason to treat different benchmarks differently.

The third assumption is that jobs execute a constant work, and that the quantity of work is the same for all jobs. Of course, on a real system, there is no reason for all jobs to execute the same quantity of work. But it is necessary to make some assumptions when defining a throughput metric. It is rare that we have any information on the representativeness of benchmarks, and we have no reason to give more weight to some benchmarks. The third assumption is fundamental: it is equivalent to saying that the number of instructions executed by a random job is not correlated with that job's IPC, i.e., we assume that, on average, a random low-IPC job executes as many instructions as a random high-IPC job.

The fourth assumption implies that the job arrival rate is sufficiently high so that throughput is limited by the machine performance, not by the arrival rate. This assumption makes sense for microarchitecture studies, where the throughput metric is used to compare different processors.

Finally, the last assumption considers that jobs are batch jobs. In general, job scheduling may impact throughput quite significantly. We also experimented with a per-core job queues and a round-robin scheduling policy, for various time quanta and varying the degree of multiprogramming: we did not observe any significant difference with FIFO scheduling. Although we have no mathematical proof, we conjecture that the second assumption (jobs are chosen randomly among benchmarks) is sufficient for throughput to be independent of the scheduling policy, *as long as the scheduling policy is unaware of jobs IPCs and treats all jobs equally.*

The metric defined by this throughput experiment will be denoted EW-IPC in the rest of the paper. Notice that, because jobs execute an equal number of instructions, the EW-IPC is directly proportional to the job throughput.

## 3.1 The IPC matrix method

A possible way to measure EW-IPC would be to run a single cycle-accurate simulation corresponding to the EW-IPC throughput experiment described in Section 3. However, it would take a very long simulation time for the EW-IPC to converge to its limit value. In practice, a faster simulation method is possible when we have few benchmarks and few cores: the *IPC matrix* method.

The basic idea is to simulate separately all the possible workloads, i.e., all the possible combinations of $K$ benchmarks, and record the threads IPCs in the IPC matrix. In the general case, there is one IPC matrix per core. Each of the $K$ matrices has one row per benchmark and one column for each combination of benchmarks on the other cores. So if we have $B$ benchmarks (or benchmarks slices) and $K$ logical cores, an IPC matrix has $B$ rows and $B^{K-1}$ columns. For instance, in the particular case of a symmetric dual-core and two benchmarks A and B, both cores have the same IPC matrix, of the form

$$\mathcal{M} = \begin{bmatrix} IPC_{AA} & IPC_{AB} \\ IPC_{BA} & IPC_{BB} \end{bmatrix}$$

where $IPC_{XY}$ is the IPC of benchmark X when running simultaneously with benchmark Y. A throughput experiment corresponding to the EW-IPC metric can be represented by an *execution trajectory*, such as the one shown in Figure 1. With 2 cores, the execution trajectory is in a 2-dimensional space: the coordinates $(x, y)$ of a point on the trajectory indicate how many instructions have been executed so far on cores 1 and 2 respectively.

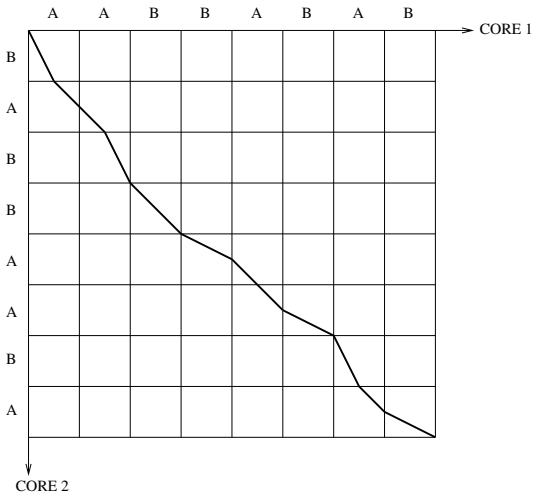



Figure 1: Example of execution trajectory on a symmetric dual-core, assuming two benchmarks A and B. Jobs of type A or B execute with equal probability. Each job executes a fixed number of instructions. At a given time, the coordinates $(x, y)$ of a point on the trajectory indicate how many instructions have been executed so far on cores 1 and 2 respectively. In this example $IPC_{BA} = 2 \times IPC_{AB}$.

Once the IPC matrices have been populated, an execution trajectory corresponding to the EW-IPC metric can be obtained with a Monte Carlo simulation: the total time can be obtained by summing the times corresponding to each segment of the trajectory. Eventually, if the Monte Carlo simulation is long enough, the total number of instructions divided by the total time provides an accurate approximation of the EW-IPC value.

The IPC matrix method is conceptually very close to the co-phase matrix method used by Van Biesbrouck et al. [15, 14].

## 3.2 A throughput paradox

EW-IPC is a more meaningful metric than commonly used equal-time throughput metrics. However, it is fundamentally different. Being closer to a realistic situation exposes the EW-IPC to the counter-intuitive behaviors that may be encountered in real situations.

To our knowledge, there is no simple analytical formula such as (1) for computing EW-IPC in the general case. In the general case, EW-IPC values must be obtained from Monte Carlo simulations. However, a simple analytical formula exists for the simple case of two benchmarks and a symmetric dual-core. We derive this formula in the rest of this section, and use it to show that the EW-IPC may actually decrease when we accelerate an individual thread.

As explained in Section 3.1, the execution trajectory is sufficient to compute the EW-IPC. The execution trajectory consists of segments. For instance, the trajectory of Figure 1 has 4 types of segments: AA, BB, AB and BA. For instance, a segment AB corresponds to core 1 running benchmark A while core 2 is running benchmark B. For such symmetric dual-core, the slope of the trajectory is equal to 1 in segments AA and BB, it is equal to $\frac{IPC_{BA}}{IPC_{AB}}$ and $\frac{IPC_{AB}}{IPC_{BA}}$ in segments AB and BA respectively.

From the execution trajectory, we can obtain the *segment fractions* $f_{AA}$, $f_{BB}$, $f_{AB}$ and $f_{BA}$

of instructions executed on segments AA,BB,AB and BA. From these segment fractions, we can obtain the total execution times $t_{AA}$, $t_{BB}$, $t_{AB}$ and $t_{BA}$ spent on segments AA, BB, AB and BA:

$$t_{AA} = \frac{f_{AA}}{2 \times IPC_{AA}} N_I \qquad t_{AB} = \frac{f_{AB}}{IPC_{AB}+IPC_{BA}} N_I$$

$$t_{BB} = \frac{f_{BB}}{2 \times IPC_{BB}} N_I \qquad t_{BA} = \frac{f_{BA}}{IPC_{AB}+IPC_{BA}} N_I$$

where $N_I$ is the total number of instructions executed. Then, the EW-IPC can be computed as

$$\text{EW-IPC} = \frac{N_I}{t_{AA}+t_{BB}+t_{AB}+t_{BA}} = \frac{1}{\frac{f_{AA}}{2 \times IPC_{AA}} + \frac{f_{BB}}{2 \times IPC_{BB}} + \frac{f_{AB}+f_{BA}}{IPC_{AB}+IPC_{BA}}} \tag{2}$$

The main difficulty lies in computing the segment fractions. In general, segment fractions must be obtained from a Monte Carlo simulation. However, in the particular case of two benchmarks executing on a symmetric dual-core, an exact analytic expression can be obtained. Notice that an execution trajectory such as the one depicted in Figure 1 does not depend on $IPC_{AA}$ and $IPC_{BB}$. Hence the following two IPC matrices

$$\mathcal{M} = \begin{bmatrix} IPC_{AA} & IPC_{AB} \\ IPC_{BA} & IPC_{BB} \end{bmatrix} \qquad \mathcal{M}' = \begin{bmatrix} IPC_{AB} & IPC_{AB} \\ IPC_{BA} & IPC_{BA} \end{bmatrix}$$

yield the exact same trajectory, hence the exact same segment fractions. But matrix $\mathcal{M}'$ corresponds to a situation where the IPC of a job does not depend on which job is running on the other core. In this situation, the two cores are independent, and the segment fractions can be computed easily (see the appendix):

$$f_{AB} = f_{BA} = \frac{1}{4} \qquad f_{AA} = \frac{1}{2} \times \frac{IPC_{BA}}{IPC_{AB}+IPC_{BA}} \qquad f_{BB} = \frac{1}{2} \times \frac{IPC_{AB}}{IPC_{AB}+IPC_{BA}} \tag{3}$$

Substituting (3) into (2), we eventually obtain the EW-IPC:

$$\text{EW-IPC} = \frac{4 \times (IPC_{AB} + IPC_{BA})}{2 + \frac{IPC_{BA}}{IPC_{AA}} + \frac{IPC_{AB}}{IPC_{BB}}} \tag{4}$$

Formula (4) is non trivial and leads to surprising consequences. In particular, the EW-IPC may decrease when we increase $IPC_{AB}$ (or symmetrically, $IPC_{BA}$):

$$\frac{IPC_{BA}}{IPC_{BB}} - \frac{IPC_{BA}}{IPC_{AA}} > 2 \Longrightarrow \frac{\partial \text{EW-IPC}}{\partial IPC_{AB}} < 0$$

For instance, consider the following two IPC matrices:

$$\mathcal{M}_1 = \begin{bmatrix} 1 & 4 \\ 2 & 0.2 \end{bmatrix} \Longrightarrow \text{EW-IPC} = 1 \qquad \mathcal{M}_2 = \begin{bmatrix} 1 & 0 \\ 2 & 0.2 \end{bmatrix} \Longrightarrow \text{EW-IPC} = 2$$

From the IPC matrices, one may believe that machine $\mathcal{M}_1$ should offer more throughput than machine $\mathcal{M}_2$, while in fact it is the opposite. In this example, decreasing $IPC_{AB}$ from 4 to 0 doubles the EW-IPC ! By freezing a job of type A when the other core runs a job of type B, we increase the job throughput, which is counter intuitive. Let us analyze what is happening. On both machines, workload BB is a workload type that we would like to avoid as much as possible because in this case the IPC is very small. On machine $\mathcal{M}_2$, a job of type A cannot terminate while the other core runs a job of type B. The only way for a workload BB to occur is when the workload running previously was also a workload BB, or when both cores start a new job

exactly at the same time (which is quite improbable). Hence on machine $\mathcal{M}_2$, workload BB is inexistent in the limit (cf. equations (3)).

This example does not mean that the EW-IPC metric is the problem. This kind of situation, where a seemingly obvious performance improvement actually decreases throughput, and which we call a *throughput paradox*, can happen in reality. However, the next section shows that throughput paradoxes are unlikely in practice.

# 4 Practical proxies for the EW-IPC

The IPC matrices corresponding to microarchitecture studies are likely to have a non-random structure. Each row of the IPC matrix gives the IPCs for one particular benchmark for all possible conditions under which this benchmark may run. Some benchmarks are "insensitive": their IPC is almost independent of the threads that are running on the other cores. Other benchmarks are "sensitive": their IPC varies more or less depending on the co-running threads.

In the particular case when all the benchmarks are insensitive, the EW-IPC can be computed easily. Let $IPC_i[b]$ be the IPC of benchmark $b$ on core $i$ (cores are not necessarily identical), which is independent of the jobs running on the other cores (as benchmarks are assumed insensitive). When we run the EW-IPC throughput experiment for a long time $T$, core $i$ executes $N_i$ instructions, and each of the $B$ benchmarks contributes nearly $N_i/B$ instructions on that core. Hence for all cores $i \in [1, K]$,

$$T = \sum_{b=1}^{B} \frac{N_i/B}{IPC_i[b]} \iff N_i = \frac{B \times T}{\sum_{b=1}^{B} \frac{1}{IPC_i[b]}}$$

The EW-IPC is $\frac{1}{T}\sum_{i=1}^{K} N_i$. Substituting $N_i$, we get

$$\text{EW-IPC} = \sum_{i=1}^{K} \frac{B}{\sum_{b=1}^{B} \frac{1}{IPC_i[b]}} \tag{5}$$

which is the sum on all cores of the harmonic mean of benchmarks IPCs.

## 4.1 Definition of the H-IPC

An interesting question to study is whether we can generalize formula (5) in the case where benchmarks are sensitive and obtain an approximation of the EW-IPC that would not necessitate Monte Carlo simulations. An obvious generalization of the harmonic mean in (5) is

$$\text{H-IPC} = \sum_{i=1}^{K} \frac{B^K}{\sum_{(b,w)} \frac{1}{IPC_i[b,w]}} \tag{6}$$

where $IPC_i[b, w]$ is the IPC value in the $b^{th}$ row and $w^{th}$ column of the IPC matrix of core $i$, and $(b, w)$ runs over all rows and columns. That is, the H-IPC is obtained by summing the harmonic means of the IPC matrices.

We distinguish two sorts of multicores: *symmetric* and *asymmetric*. We say that a multicore is symmetric when all the cores have equivalent IPC matrices[3]. A multicore that is not symmetric is asymmetric. For instance, most general-purpose multicore processors today are symmetric, including those whose physical cores are SMT.

[3] The IPC matrices of two cores are equivalent if the two matrices can be made equal by a renumbering of cores

$$IPC[b,w] = \min(3, \max(0.1, BIPC[b] \times S[b,w]))$$
Random variable $S[b,w]$ normally distributed with mean equal to 1
$$BIPC[b] = 0.1 + R[b] \times 2.9 \quad \text{for } b \in [1,B]$$
Random variable $R[b]$ uniform in $[0,1]$

Figure 2: IPC matrix model

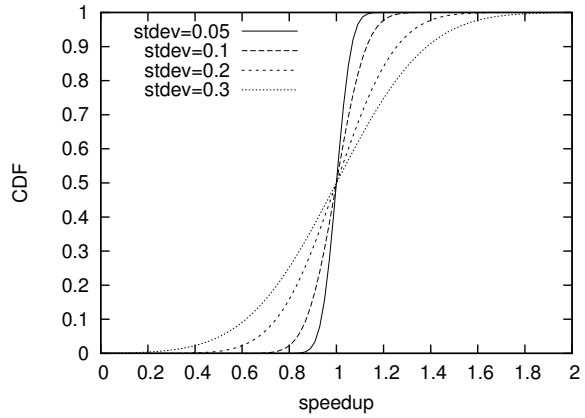



Figure 3: IPC matrix model: cumulative distribution function of the normally-distributed speedup $S[b,w]$ for different values of the standard deviation STDEV.

For symmetric multicores, the H-IPC formula can be simplified:

$$\text{H-IPC} = \frac{K \times B^K}{\sum_{(b,w)} \frac{1}{IPC[b,w]}} \tag{7}$$

Unlike the EW-IPC, the H-IPC is not associated with a throughput experiment and its physical meaning can only be approximate. Nevertheless, we conjecture that the H-IPC can be used as a proxy for the EW-IPC in most practical situations. Ideally, we would like to check this conjecture by doing many different microarchitecture studies and by checking that the EW-IPC and the H-IPC lead to the same conclusions. However, we would be limited in practice to a few microarchitecture studies with small IPC matrices, from which it would be difficult to conclude anything or gain any insight. Instead, we use the artificial IPC matrix model described in Figure 2. In this model, $S[b,w]$ represents a speedup following a normal distribution of mean 1 and whose standard deviation STDEV is a parameter that we will vary. For symmetric multicores, we enforce $S[b,w_1] = S[b,w_2]$ for workloads $w_1$ and $w_2$ that are actually the same benchmarks multiset.

STDEV quantifies benchmarks sensitivity: the higher STDEV, the more sensitive the benchmarks. Figure 3 shows the cumulative distribution function of $S[b,w]$ for different values of STDEV. For instance, with STDEV=0.3, there is a probability of about 50% that the speedup is less than 0.8 or greater than 1.2. In the context of microarchitecture studies, this corresponds to very sensitive benchmarks.

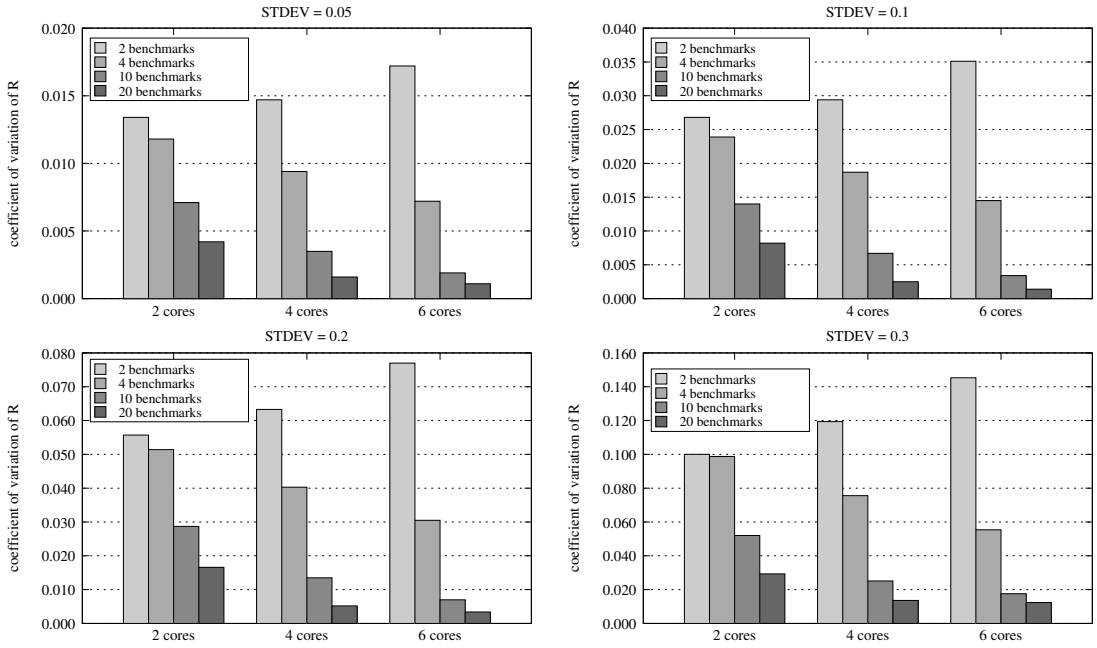



Figure 4: Coefficient of variation (lower is better) of $R = \frac{\text{H-IPC}}{\text{EW-IPC}}$ for symmetric multicores, for different values of STDEV.

Our goal is to understand to what extent the H-IPC can be used as a proxy for the EW-IPC. This does not necessarily mean that the H-IPC values should be close to the EW-IPC values. The H-IPC and EW-IPC are *equivalent* if, for any two machines X and Y,

$$\frac{\text{H-IPC}[Y]}{\text{H-IPC}[X]} \approx \frac{\text{EW-IPC}[Y]}{\text{EW-IPC}[X]} \tag{8}$$

In words, the two metrics must indicate approximately the same speedup. Condition (8) is equivalent to

$$\frac{\text{H-IPC}[Y]}{\text{EW-IPC}[Y]} \approx \frac{\text{H-IPC}[X]}{\text{EW-IPC}[X]} \tag{9}$$

That is, the H-IPC and EW-IPC are equivalent if the ratio $R = \frac{\text{H-IPC}}{\text{EW-IPC}}$ is approximately constant (not necessarily equal to 1). To quantify the equivalence between the two metrics, we generate 1000 IPC matrices according to the IPC matrix model of Figure 2. For each IPC matrix, we compute $R$ (we obtain the EW-IPC with Monte Carlo simulations). Eventually, the coefficient of variation $CV_R = \sigma_R/\mu_R$ (with $\mu_R$ and $\sigma_R$ respectively the mean and standard deviation of $R$) quantifies the extent to which the two metrics are equivalent. We observed experimentally that the fraction $R \in [\mu_R - 2\sigma_R, \mu_R + 2\sigma_R]$ is greater than 0.93. For instance, a $CV_R$ of 1% means that if we compare two microarchitectures X and Y, and if the H-IPC of Y is 10% higher than the H-IPC of X, then the EW-IPC of Y is likely between 8% and 12% higher than the EW-IPC of X.

Results are presented in Figure 4 for symmetric multicores and in Figure 5 for asymmetric multicores. These graphs show $CV_R$ when we vary the number of cores $K$, the number of

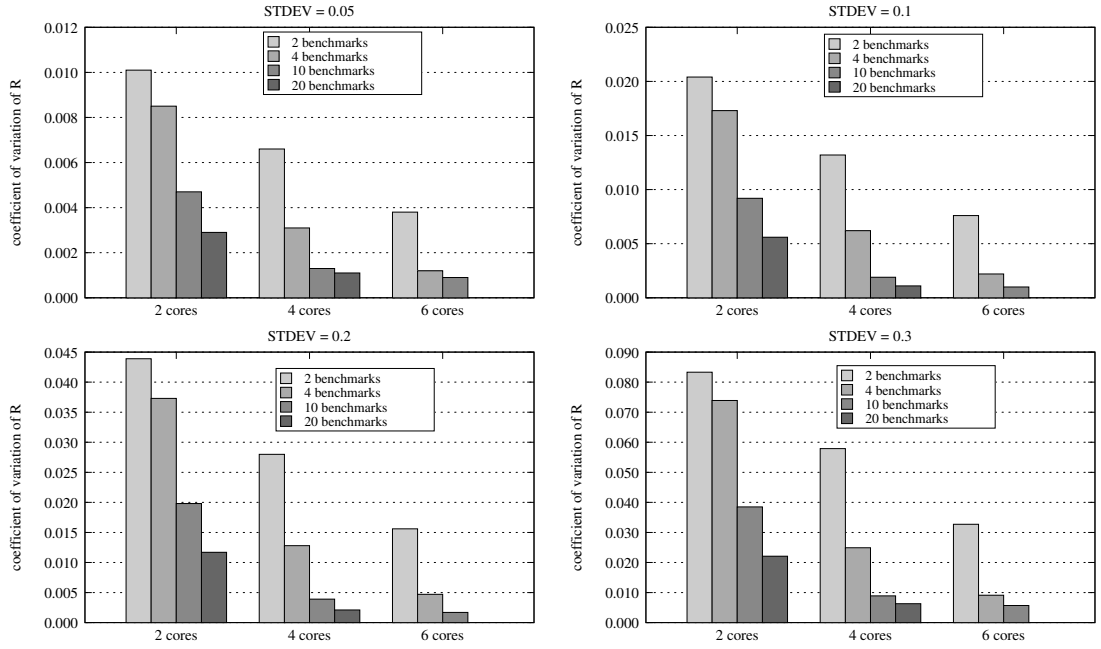



Figure 5: Coefficient of variation (lower is better) of $R = \frac{\text{H-IPC}}{\text{EW-IPC}}$ for asymmetric multicores, for different values of STDEV.

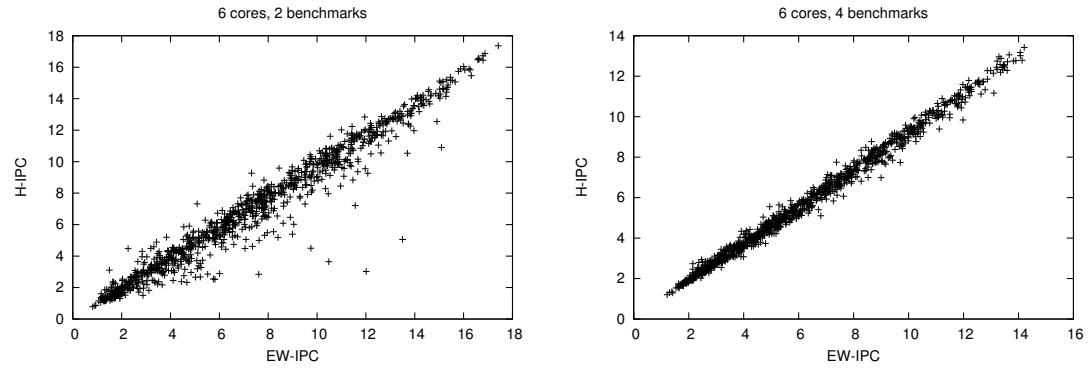



Figure 6: Scatter plots with the EW-IPC on the x axis and the H-IPC on the y axis (1000 points), for a symmetric 6-core, assuming STDEV=0.3. The left plot is for 2 benchmarks, and the right plot for 4 benchmarks.

benchmarks $B$, and the IPC matrix model STDEV[4]. Figure 6 shows some scatter plots for a couple of cases (symmetric 6-core, STDEV=0.3, 2 and 4 benchmarks). Several observations can be made, some of which not obvious: As expected, $CV_R$ increases with STDEV, that is, the more sensitive the benchmarks, the greater the discrepancy between the H-IPC and the EW-IPC; $CV_R$ decreases when the number of benchmarks is increased; Except for symmetric multicores with 2 benchmarks, $CV_R$ decreases when we increase the number of cores; $CV_R$ is lower for asymmetric multicores.

In practice, the H-IPC can be used as a throughput metric in microarchitecture studies, in lieu of the harder-to-obtain EW-IPC. For instance, if we use 20 benchmarks to study a symmetric dual-core, and if the benchmarks are moderately sensitive (STDEV $< 0.1$), $CV_R$ is less than 1% and the H-IPC can reasonably be used to establish throughput differences of a few percents. Still, what we show with the H-IPC is a performance under workloads involving several different behaviors. Under certain workloads, in particular workloads involving a small number of distinct applications with a "monolithic" behavior, or applications with similar behaviors, the observed performance might be quite different from the conclusions obtained on the whole benchmark suite (cf. throughput paradoxes, Section 3.2). It would be possible to show several throughput numbers, one throughput number for the whole benchmark suite, and also throughput numbers for subsets of benchmarks, e.g., benchmarks pairs. However, the H-IPC is not a safe metric for studying sensitive benchmark pairs, as it might lead to conclusions having no physical reality.

## 4.2 Sampling the IPC matrix: the ASH-IPC and SSH-IPC

So far, we have assumed that the IPC matrix is fully populated. However, in practical microarchitecture studies, it is often not possible to fill the IPC matrix completely because this would require too many simulations. A frequent practice is to simulate a sample of several tens, sometimes a few hundreds of different workloads. A sampled version of the H-IPC can be defined as follows. Let $W$ be the number of workloads in the sample and let $IPC(w,i)$ be the IPC of the thread on core $i$ in the $w^{th}$ workloads. We define the *asymmetric sampled* H-IPC, denoted ASH-IPC, as

$$\text{ASH-IPC} = \sum_{i=1}^{K} \frac{W}{\sum_{w=1}^{W} \frac{1}{IPC(w,i)}} \tag{10}$$

That is, the ASH-IPC is the sum of per-core harmonic means of IPCs. In the case of a symmetric multicore, the IPCs of all cores come from the same IPC matrix, and we can use the *symmetric sampled* H-IPC, denoted SSH-IPC:

$$\text{SSH-IPC} = K \times \frac{W \times K}{\sum_{i=1}^{K} \sum_{w=1}^{W} \frac{1}{IPC(w,i)}} \tag{11}$$

That is, the SSH-IPC is the global harmonic mean of IPCs times the number of cores.

The coefficient of variation of $\frac{\text{ASH-IPC}}{\text{H-IPC}}$ and $\frac{\text{SSH-IPC}}{\text{H-IPC}}$ quantifies the extent to which the ASH-IPC and SSH-IPC give the same conclusions as the H-IPC. As in Section 4.1, we define 1000 IPC matrices according to the model of Figure 2, for symmetric and asymmetric multicores.

Results are shown in Figures 7 and 8 for a symmetric and asymmetric 4-core respectively, assuming 20 benchmarks, STDEV=0.05 and STDEV=0.3. The left graphs show the coefficient of variation of $\frac{\text{ASH-IPC}}{\text{H-IPC}}$ as a function of the number of workloads in the sample, and the right

[4]For the asymmetric 6-core case, we do not have results for 20 benchmarks because the IPC matrices are very big, which makes simulations very slow. For symmetric multicores, we use a much more compact representation of the IPC matrix, taking advantage of the redundancies in it.

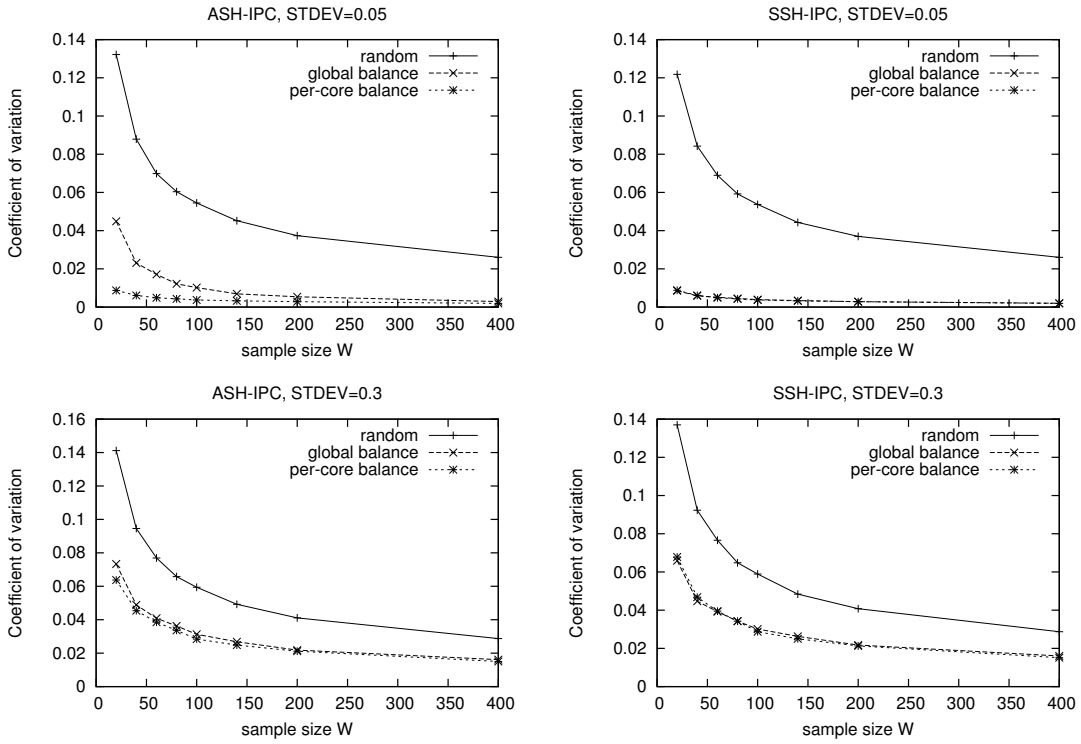



Figure 7: Coefficient of variation of $\frac{\text{ASH-IPC}}{\text{H-IPC}}$ (left graphs) and $\frac{\text{SSH-IPC}}{\text{H-IPC}}$ (right graphs) for a symmetric 4-core, assuming 20 benchmarks, STDEV=0.05 and STDEV=0.3.

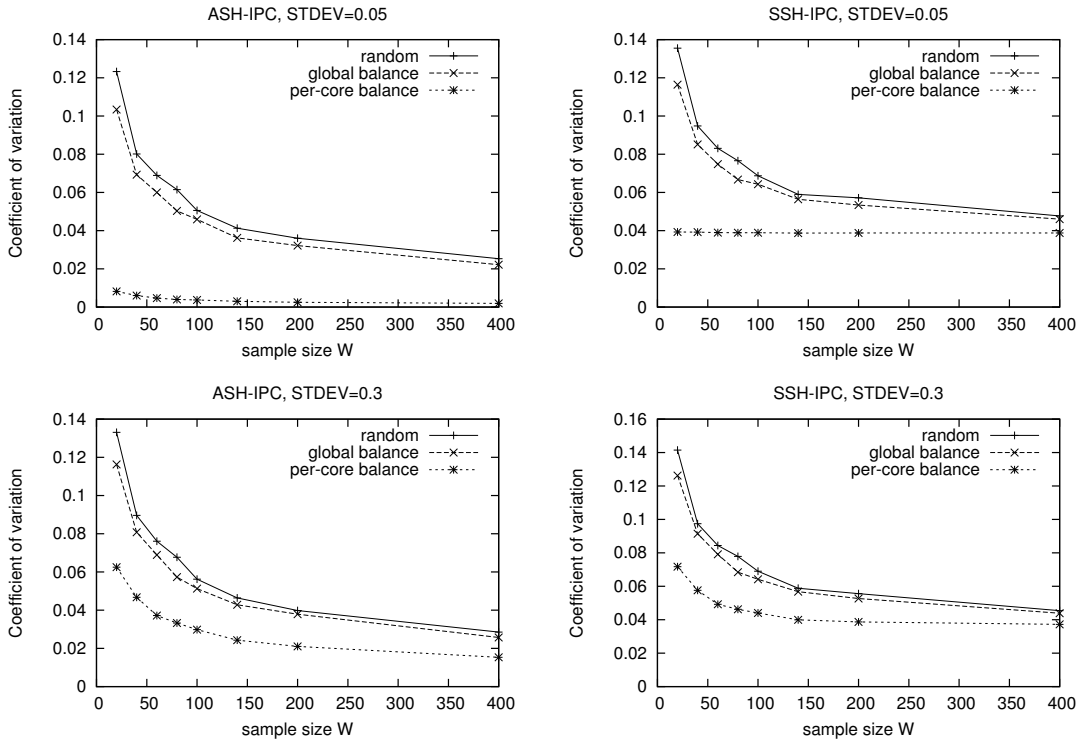



Figure 8: Coefficient of variation of $\frac{\text{ASH-IPC}}{\text{H-IPC}}$ (left graphs) and $\frac{\text{SSH-IPC}}{\text{H-IPC}}$ (right graphs) for an asymmetric 4-core, assuming 20 benchmarks, STDEV=0.05 and STDEV=0.3.

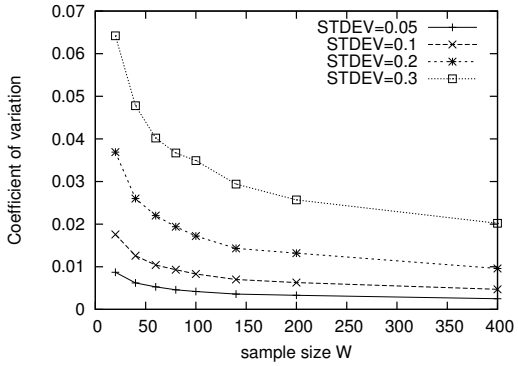



Figure 9: Coefficient of variation of $S = \frac{\text{ASH-IPC}}{\text{EW-IPC}}$ for a symmetric 4-core , with 20 benchmarks, using per-core balanced workloads.

graphs show the coefficient of variation of $\frac{\text{SSH-IPC}}{\text{H-IPC}}$. For each graph, we show 3 curves, each corresponding to a different way to select the workloads:

- **Random**. Workloads are chosen completely randomly.
- **Global balance**. Among the total $W \times K$ threads constituting the $W$ workloads, each benchmark occurs $K \times W/B$ times, $W$ being a multiple of the number $B$ of benchmarks.
- **Per-core balance**. Each benchmark occurs $W/B$ times *on each core*, with $W$ a multiple of $B$.

A first observation is that balancing the benchmarks decreases significantly the coefficient of variation. For asymmetric multicores, per-core balance is the best of the 3 workload selection methods. For symmetric multicores, per-core balance is better than global balance for the ASH-IPC. For symmetric multicores, the SSH-IPC is nearly equivalent to the ASH-IPC under per-core balance, and it is better than the ASH-IPC under global balance and under random workloads. For asymmetric multicores, as expected, the ASH-IPC is superior to the SSH-IPC. In conclusion, we recommend using per-core balanced workload samples and the ASH-IPC or, for symmetric multicores, the SSH-IPC.

The required sample size depends on benchmarks sensitivity and on the microarchitectures being compared. If benchmarks have a small sensitivity (STDEV $< 0.05$) and if the performance difference between the two microarchitectures is significant (a few percents), taking a few tens of workloads is ok. But if the performance difference between the microarchitectures is less than 1%, several hundreds of workloads may be necessary to quantify precisely the performance difference. Figure 9 shows the coefficient of variation $CV_S$ of $S = \frac{\text{ASH-IPC}}{\text{EW-IPC}}$ for a symmetric 4-core with 20 benchmarks, using per-core balanced workloads. The number of workloads necessary to make $CV_S$ 1% or smaller increases with benchmarks sensitivity (i.e., with STDEV). For STDEV=0.05, 20 workloads are sufficient for having $CV_S \lesssim 0.01$, but we need 60 workloads for STDEV=0.1 and 400 workloads for STDEV=0.2.

| workload | single-thread reference IPC | IPC SoE fairness=0.25 | IPC SoE fairness=1 |
|---|---|---|---|
| mgrid+art | 0.31 ; 0.42 | 0.28 ; 0.19 | 0.25 ; 0.19 |
| lucas+applu | 0.78 ; 0.75 | 0.56 ; 0.50 | 0.56 ; 0.50 |
| galgel+gcc | 0.22 ; 1.31 | 0.06 ; 1.03 | 0.11 ; 0.78 |
| vortex+gzip | 1.03 ; 1.36 | 0.25 ; 1.03 | 0.56 ; 0.67 |
| gcc+eon | 0.56 ; 1.44 | 0.08 ; 1.25 | 0.33 ; 0.92 |
| gap+vpr | 1.36 ; 1.19 | 1.11 ; 0.31 | 0.78 ; 0.56 |
| vpr+eon | 1.19 ; 1.72 | 0.22 ; 1.33 | 0.58 ; 0.83 |
| apsi+swim | 2.06 ; 0.28 | 1.81 ; 0.06 | 1.31 ; 0.19 |
| mgrid+mgrid | 0.61 ; 0.61 | 0.50 ; 0.50 | 0.47 ; 0.44 |
| bzip2+bzip2 | 1.92 ; 1.92 | 1.08 ; 1.03 | 1.06 ; 1.06 |

Table 2: IPC numbers reproduced approximately from Figure 6 in reference [3]

# 5 Implications for the microarchitect

From a mathematical point of view, the difference between an equal-time throughput metric such as the ET-IPC and an equal-work throughput metric such as the H-IPC is the difference between an arithmetic mean and a harmonic mean. Microarchitecture techniques that increase all threads IPCs will likely be assessed similarly by the ET-IPC and the H-IPC, at least qualitatively. However the situation is more complex when some IPCs are increased while others are decreased, which happens for instance when comparing several different ways to arbitrate shared resources. Using the ET-IPC favors solutions that make high IPCs higher, even if this makes low IPCs lower. In particular, if a high-IPC thread is in conflict with a low-IPC thread for a resource, the ET-IPC tends to favor solutions that give priority to the high-IPC thread. Contrary to the ET-IPC, the H-IPC favors solutions that tend to equalize the threads IPCs. That is, when a high-IPC thread is in conflict for a resource with a low-IPC thread, the H-IPC tends to favor solutions that give priority to the low-IPC thread.

The following example is taken from a study by Gabor et al. published in 2006 [3]. In this study, the authors propose a mechanism for controlling fairness in processors implementing Switch-on-Event (SoE) multithreading, aka coarse grained multithreading. With SoE multithreading, a single thread is running at a given time, the other threads are waiting. Upon a long latency event, like a last-level cache miss, the execution switches to another thread. This permits hiding (to some exent) the cache miss latency. However, the authors show that a naive implementation of SoE that switches threads on every cache miss may be very unfair to small-IPC threads that execute few instructions between consecutive misses. Therefore, they propose a mechanism for controlling fairness. The relative performance of a thread is the thread's IPC in SoE mode divided by the thread's IPC when running alone. They define fairness as the minimum relative performance divided by the maximum relative performance. In the proposed mechanism, the fairness target is set between 0 (no fairness enforcement) and 1 (strong fairness enforcement), and the mechanism introduces extra thread switches in such a way that the actual fairness is close to the fairness target. One of the conclusions they draw from their experiments is that enforcing fairness requires to sacrifice some throughput. But the throughput metric they used is the ET-IPC.

Table 2 shows the IPC number that we have measured approximately from Figure 6 in reference [3], which gives the instruction throughput obtained by simulating a dual-threaded

| | ET-IPC | weighted speedup | harmonic mean of speedups | ASH-IPC | SSH-IPC |
|---|---|---|---|---|---|
| fairness=0.25 | 1.32 | 1.17 | 0.41 | 0.51 | 0.49 |
| fairness=1 | 1.22 | 1.19 | 0.58 | 0.82 | 0.81 |
| ratio | 0.92 | 1.02 | 1.40 | 1.61 | 1.65 |

Table 3: Throughput metrics applied on the IPC numbers of Table 2

SoE processor, for different values of the fairness target[5]. We give in Table 3 the throughput numbers for several metrics applied on the IPCs of Table 2. According to the ET-IPC (second column), increasing the fairness target decreases throughput moderately, which is consistent with the conclusions of [3]. The authors of [3] wanted to distinguish clearly notions of throughput and fairness, this is why they did not use the weighted speedup and the harmonic mean of speedups. Nevertheless, we show these two metrics in the third and fourth columns since they are still frequently used. Going from a fairness target of 0.25 to a fainess target of 1 keeps weighted speedup almost the same. However, a fairness target of 1 increases the harmonic mean of speedups by roughly 40% compared to a fairness target of 0.25. The last two columns of Table 3 give the ASH-IPC and the SSH-IPC. According to these equal-work throughput metrics, a fairness target of 1 yields about 60%-65% more throughput than a fairness target of 0.25. Equal-work throughput is increased because enforcing fairness increases the fraction of processor time allotted to small-IPC threads, thereby increasing these threads IPCs. Had the authors of [3] used the ASH-IPC or the SSH-IPC, they would have concluded that not only does their mechanism allow to control fairness, but it increases throughput quite substantially.

# 6 Summary and conclusion

The throughput metrics used in microarchitecture sudies so far are equal-time throughput metrics based on the assumption that all the jobs execute for a fixed and equal time. We argue that this assumption is not realistic, and we advocate for equal-work throughput metrics based on the assumption that the IPC of a random job is not correlated with the total number of instructions executed by that job. We have introduced such equal-work throughput metric, called the EW-IPC, which is associated with a throughput experiment and hence has a clear physical meaning. We have shown that the EW-IPC could lead to throughput paradoxes such that accelerating an individual thread might decrease throughput. Because there is no simple analytical formula for the EW-IPC in the general case, we introduced the H-IPC, a simple proxy for the EW-IPC. Using an IPC matrix model, we have studied to what extent the H-IPC is a good proxy for the EW-IPC. We have considered the practical case where only a relatively small number of workloads are simulated, and we have proposed two different estimators for the H-IPC in this case, the ASH-IPC and the SSH-IPC. We have shown that these estimators work best when all the benchmarks are equally represented in the sample workloads. We have shown with an example that using an equal-work throughput metric such as the H-IPC may lead to conclude differently in a microarchitecture study. While equal-time throughput metrics tend to favor microarchitectures that accelerate faster threads, equal-work throughput metrics tend to favor microarchitectures that make threads performance more uniform. In conclusion, we recommend

[5]Figure 6 in [3] shows only 10 of the 16 workloads they have used for their study. The missing workloads are homogeneous workloads that are slightly impacted by fairness enforcement.

that microarchitecture studies concerned with multiprogram throughput use the ASH-IPC or, for symmetric multicores, the SSH-IPC.

# Appendix

Let us assume that the IPC of a running job does not depend on the job running on the other core. Consequently, the two cores are independent. We denote $IPC_A$ and $IPC_B$ the IPCs of jobs of type A and B respectively. Let $P_A$ and $P_B = 1 - P_A$ be the probabilities that, at a random time, a given core is executing a job of type A or B respectively. In the limit, because of the assumptions for the EW-IPC (Section 3), a given core executes as many instructions from jobs of type A and B, which means

$$\begin{aligned} P_A \times IPC_A &= P_B \times IPC_B \\ &= (1 - P_A) IPC_B \end{aligned}$$

Solving this equation for $P_A$, we get

$$P_A = \frac{IPC_B}{IPC_A + IPC_B} \tag{12}$$

$$P_B = \frac{IPC_A}{IPC_A + IPC_B} \tag{13}$$

The average total IPC is

$$IPC = 2 \times (P_A \times IPC_A + P_B \times IPC_B) = \frac{4 IPC_A IPC_B}{IPC_A + IPC_B} \tag{14}$$

Because the two cores are independent, the probabilities can be multiplied. For instance, the probability to be on a segment of type AB at a random time (that is, core 1 is executing a job of type A while core 2 is executing a job of type B) is equal to $P_A \times P_B$. The segment fractions are:

$$\begin{aligned} f_{AA} &= \frac{P_A^2 \times 2 \times IPC_A}{IPC} \\ f_{BB} &= \frac{P_B^2 \times 2 \times IPC_B}{IPC} \\ f_{AB} = f_{BA} &= \frac{P_A P_B \times (IPC_A + IPC_B)}{IPC} \end{aligned}$$

Replacing $P_A$, $P_B$ and $IPC$ with the expressions in equations (12), (13) and (14), yields

$$\begin{aligned} f_{AB} = f_{BA} &= \frac{1}{4} \\ f_{AA} &= \frac{1}{2} \times \frac{IPC_B}{IPC_A + IPC_B} \\ f_{BB} &= \frac{1}{2} \times \frac{IPC_A}{IPC_A + IPC_B} \end{aligned}$$

# References

[1] A. Carlton. CINT92 and CFP92 homogeneous capacity method offers fair measure of processing capacity. http://www.spec.org/cpu92/specrate.txt.

[2] P. J. Fleming and J. J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, Mar. 1986.

[3] R. Gabor, S. Weiss, and A. Mendelson. Fairness and throughput in switch on event multithreading. In *Proc. of the 39th Annual International Symposium on Microarchitecture*, 2006.

[4] M. F. Iqbal and L. K. John. Confusion by all means. In *Workshop on Unique Chips and Systems (UCAS)*, 2010.

[5] J. R. Mashey. War of the benchmark means : Time for a truce. *ACM SIGARCH Computer Architecture News*, 32(4), Sept. 2004.

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

[7] S. Parekh, S. Eggers, H. Levy, and J. Lo. Thread-sensitive scheduling for SMT processors. Technical Report UW-CSE-00-04-02, University of Washington, 2000.

[8] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proc. ot the International Symposium on Microarchitecture*, 2006.

[9] Y. Sazeides and T. Juan. How to compare the performance of two SMT microarchitectures. In *Proc. of the IEEE International Symposium on Performance Analysis of Software and Systems*, 2001.

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

[11] A. Snavely, D. M. Tullsen, and G. Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *Proc. of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2002.

[12] D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, and R. L. Stamm. Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proc. of the 23rd Annual International Symposium on Computer Architecture*, 1996.

[13] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading : Maximizing on-chip parallelism. In *Proc. of the 22nd Annual International Symposium on Computer Architecture*, 1995.

[14] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Considering all starting points for simultaneous multithreading simulation. In *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2006.

[15] M. Van Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *Proc. of the IEEE International Symposium on Performance Analysis of Systems and Software*, 2004.


**RESEARCH CENTRE**
**RENNES – BRETAGNE ATLANTIQUE**

Campus universitaire de Beaulieu
35042 Rennes Cedex

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

ISSN 0249-6399