Defining metrics for multicore throughput on multiprogrammed workloads

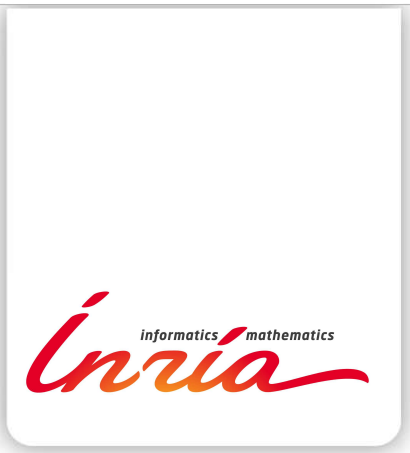
Stijn Eyerman, Pierre Michaud

Defining metrics for multicore throughput on multiprogrammed workloads

Stijn Eyerman, Pierre Michaud

RESEARCH REPORT
N° 8401
November 2013
Project-Team ALF

ISSN 0249-6399 ISRN INRIA/RR--8401--FR+ENG



Defining metrics for multicore throughput on multiprogrammed workloads

Stijn Eyerman*, Pierre Michaud

Project-Team ALF

Research Report n° 8401 — November 2013 — 23 pages

Abstract: Measuring throughput is not as straightforward as measuring execution time. This has led to an ongoing debate on what forms a meaningful throughput metric for multi-program workloads. We present a method to construct throughput metrics in a systematic way: we start by expressing assumptions on job size, job distribution, scheduling, etc., that together define a theoretical throughput experiment. The throughput metric is then the average throughput of this experiment. Different assumptions lead to different metrics, so one should select the metric whose assumptions are close to the real usage he/she has in mind. We elaborate multiple metrics based on different assumptions. In particular, we identify the assumptions that lead to the commonly used weighted speedup and harmonic mean of speedups. Our study clarifies that they are actual throughput metrics, which was recently questioned. We also propose some new throughput metrics, whose calculation sometimes requires approximation. We use synthetic and real experimental data to characterize metrics and show how they relate to each other. Our study can also serve as a starting point if one needs to define a new metric based on specific assumptions, other than the ones we consider in this study. Throughput metrics should always be defined from explicit assumptions, because this leads to a better understanding of the implications and limits of the results obtained with that metric.

Key-words: Throughput metric, multicore processor, multi-programmed workload

* Ghent University

RESEARCH CENTRE
RENNES – BRETAGNE ATLANTIQUE

Campus universitaire de Beaulieu
35042 Rennes Cedex

Définition de grandeurs de débit d'exécution de charges multi-programmées sur les processeurs multi-cœurs

Résumé : Il est plus difficile de mesurer le débit d'exécution d'un processeur que le temps d'exécution d'une tâche seule. Ceci explique le débat actuel concernant la bonne manière de mesurer le débit d'exécution sur des charges multi-programmées. Nous présentons une méthode permettant de définir de manière systématique des grandeurs de débit d'exécution. Nous commençons par poser des hypothèses sur la taille des tâches, la distribution de leurs caractéristiques, l'algorithme de planification, etc., qui ensemble définissent complètement une expérience théorique de mesure de débit. La grandeur de débit ainsi définie correspond au débit moyen mesuré sur cette expérience. Différentes hypothèses conduisent à des grandeurs de débit différentes. L'utilisateur doit logiquement choisir les grandeurs de débit dont les hypothèses sont les plus proches du cas pratique visé par son étude. Nous présentons plusieurs grandeurs de débit basées sur différentes hypothèses. En particulier, nous identifions les hypothèses sous-jacentes à deux grandeurs de débit fréquemment utilisées: l'accélération pondérée et la moyenne harmonique des accélérations. Notre étude établit que ce sont effectivement des grandeurs de débit, un fait récemment contesté. Nous proposons de nouvelles grandeurs de débit, l'évaluation de certaines d'entre elles nécessitant des formules approchées. Nous caractérisons ces grandeurs de débit et comparons leurs comportements respectifs sur des données artificielles ou empiriques. Notre étude peut servir de modèle pour des utilisateurs devant définir de nouvelles grandeurs de débit basées sur des hypothèses spécifiques, autres que celles que nous avons considérées. Les grandeurs de débit devraient toujours être définies à partir d'hypothèses explicites, car cela donne une meilleur compréhension des implications et des conditions de validité des résultats.

Mots-clés : Grandeur de débit, processeur multi-cœur, charges multi-programmées

1 Introduction

Metrics are at the foundation of experimental evaluation. Choosing the correct metric is as important as setting up a meaningful experiment. The choice of the metric should reflect the goal of the study as closely as possible. For example, a cache optimization study can report cache miss rate reductions, but if the end goal is to improve performance, run time reductions are more meaningful.

Computer architecture research is complicated by the diversity of application domains and specific user settings, making it almost impossible to define a standard benchmarking setup that can be used by all researchers. Benchmark suites are supposed to represent typical behavior in a certain application domain, but there is usually no information about the relative importance of each of the benchmarks and their typical run time, because this depends on the specific setting and inputs. In general, we expect benchmarks to be representative of behaviors exhibited by real workloads, but the benchmarks themselves are not real workloads.

Evaluating performance for *single-program* workloads (single- or multi-threaded) is straightforward: the end goal is to reduce the run time. On *multi-program* workloads, though, the goal is not to decrease the run time of individual programs, but to increase throughput. Throughput is less strictly defined than run time, leading to many possible interpretations. As a result, there is an ongoing debate on what forms a good throughput metric for multi-program workloads [2, 4, 10, 11, 15].

Throughput is usually defined as the number of jobs executed per unit of time. The exact throughput number depends on many characteristics of the setup: job arrival times, job behaviors, job sizes, scheduling policy, etc. Ideally, throughput should be measured using a *throughput experiment* (TPEX): a simulation setup where jobs arrive, are scheduled, executed, and leave the system, using representative arrival times, job burst modeling, job lengths, job distribution, etc. The problem is that it is not easy to set up an actual TPEX and it is a challenge to perform sufficient representative experiments, due to the large number of possible settings.

A common practice in microarchitecture studies using multi-program workloads is to select a sample of benchmark combinations, and simulate each sample separately. Typically, the number of benchmarks in a sample does not exceed the maximum number of threads that can run in parallel, to avoid modeling context switches and scheduling. In the end, the results of these simulations should be averaged into a throughput number that is equal to that of an actual TPEX. This averaging is not uniquely defined, because it relies on assumptions that may impact the final throughput.

In this paper, we present a method to construct a meaningful throughput metric to calculate throughput without doing an actual throughput experiment. We elaborate some new metrics and give a new meaning to known metrics. Because an absolute throughput number depends on many assumptions, there is no single throughput metric. In fact, a particular study can require constructing a metric that is specific to that study only. Furthermore, different assumptions lead to different metrics, which can result in contradictory conclusions. However, it is important to have a systematic method to construct metrics, such that *we can choose a metric that is consistent with the goal of a study, instead of using an arbitrary metric with an unclear meaning.*

The starting point for constructing a metric is to define a theoretical TPEX. The TPEX is defined by assumptions about job sizes, job distribution, scheduling, etc., such that all assumptions completely determine an experiment that could be done in reality. The throughput metric is then the throughput measured on this TPEX. This means that a metric is defined by the assumptions of the TPEX and not by a mathematical formula. A mathematical formula can be derived from the assumptions, forming a practical way to calculate throughput. However, we

will see that it is not always possible to construct an exact formula for each TPEX, in which case we have to use approximative formulas for these metrics.

The paper is organized as follows. Section 2 provides an overview of the related work. Section 3 shows how to calculate instantaneous throughput, i.e., the throughput at a certain time, while Section 4 elaborates average throughput metrics. Section 5 discusses practical considerations for using these metrics in a constrained simulation setup. Sections 6 and 7 analyze the impact of choosing a metric. We finally discuss future work and conclude in Sections 8 and 9. In this paper, we derive multiple metrics that are new or give a new meaning to existing metrics. However, we would like to stress that the example metrics in this paper are not meant to be a restrictive or comprehensive list. Nevertheless, we believe that the example metrics are constructed using as general and as fair assumptions as possible, making them useful for a wide range of studies.

2 Background

Throughput metrics are used to evaluate the quantity of work that a machine can do in a given time. Some computer architecture studies actually simulate real TPEXs (throughput experiments) [15, 13, 8, 12, 3]. However, studies focusing on microarchitecture generally avoid TPEXs because of the long simulation time. Instead, these studies usually consider *instantaneous throughput*.

2.1 Instantaneous throughput for computer architecture studies

The instantaneous throughput of a machine is the quantity of work done during a short time window. It is defined at a given instant for the current *coschedule*, i.e., the particular set of jobs that are running concurrently at that time. The instantaneous throughput for an arbitrary coschedule can be obtained by simulation. Instantaneous throughput does not tell how representative a coschedule is, but it provides insight on how well particular combinations of benchmarks run together.

Different instantaneous throughput metrics can be defined depending on the choice of a *unit of work* (UoW). Early studies on simultaneous multithreading (SMT) used the instruction as the UoW [18, 17] and report total *IPC throughput* [2]. However, IPC throughput is problematic under heterogeneous workloads. When a high-IPC job and a low-IPC job execute simultaneously, a microarchitecture maximizing IPC throughput would have resource arbitration policies advantaging the high-IPC job, which may be unfair to the low-IPC one.

Several researchers independently proposed the *weighted speedup* (WSU) metric for avoiding the bias toward high IPCs [15, 13, 14]. WSU is the sum of the speedups of the coexecuting jobs, where speedup is the acceleration over a reference core. Here, the quantity of work per instruction differs per benchmark, it is proportional to the benchmark's average CPI (cycles per instruction) on the reference core.

The *harmonic mean of speedups* (HSU) is also frequently used in computer architecture studies. Like WSU, it is based on the speedups of coexecuting jobs, but instead of summing the speedups, we take their harmonic mean. HSU was proposed by Luo et al. as a way to mix in the same performance formula notions of throughput and fairness [10].

2.2 Measuring instantaneous throughput

Instantaneous throughput can be measured by simulating the execution of a coschedule of K jobs for a fixed time, with K the number of cores[1]. A possible method simulates the K jobs and measures throughput during a short time window. However the work actually executed during the time window depends on the machine's performance. Microarchitects generally prefer to compare different machines on the same work [14, 6]. To solve this issue, Hilton et al. proposed to fix the work on which two machines are compared, i.e., each job executes a fixed number of instructions [6]. When a job is finished, the core on which it executed remains idle. To minimize the "tail" effect, i.e., the idle time at the end of the simulation, Hilton et al. propose to dimension job sizes so that all jobs have the same run time when executed alone. Another solution is to re-execute a job as soon as it is finished, multiple times if necessary, so that the tail effect is negligible [16, 25].

2.3 Average throughput

Instantaneous throughput characterizes a particular coschedule at a particular point in time. In general, we characterize the throughput of a machine by averaging its instantaneous throughput over several different coschedules. Several questions arise: Are all coschedules equally important? How to select the coschedules? Which sort of averaging should be done? What about the phase behavior of programs?

Concerning the phase behavior of programs, Kihm et al. [7] and Van Biesbrouck et al. [19] argue that all the possible offsets between coexecuting jobs should be considered equally likely. They propose methods that yield an average throughput value for one particular set of coexecuting jobs.

In many microarchitecture studies, the authors "manually" select coschedules that are presumably interesting for the case considered. Yet, we do simulations precisely because the microarchitectural behavior of programs is complex and not easily grasped by intuition. Hence researchers have looked for systematic methods. Most of these methods assume that all the benchmarks are equally important. Van Biesbrouck et al. classify co-phases based on microarchitecture-independent characteristics, and they use clustering to identify a set of representative co-phases [20]. Random sampling, i.e., randomly chosen coschedules, is also often used. Van Craeynest and Eeckhout show that, if we define average throughput on the full population of possible coschedules, it may be more accurate to use a large random sample and fast approximate simulation than a small sample of manually chosen coschedules and detailed simulation [22]. Velásquez et al. show that fast approximate simulations may help select a representative sample of coschedules for detailed simulations [24].

2.4 The meaning of throughput metrics

WSU and HSU were invented in an intuitive manner. A few studies have questioned the meaning of these metrics. The meaning of WSU as an instantaneous throughput metric has been explained by Eyerman and Eeckhout [2]. HSU was not introduced as a throughput metric but as a mathematical formula that mixes notions of throughput and fairness. Literally, HSU is the inverse of the average slowdown experienced by a job when coexecuting with some other jobs [2].

Vandierendonck and Seznec show experimentally that WSU and HSU tend to give similar conclusions [23]. This confirms an empirical fact: published studies that use both metrics often obtain similar qualitative conclusions from them.

[1] In the remaining, the term "core" denotes a virtual core, i.e., a hardware context. An SMT physical core represents several virtual cores.

A debate was recently started by Michaud [11]. Michaud argues that WSU and HSU are inconsistent in the sense that the conclusions obtained may change dramatically depending on the chosen reference core. He advocates the use of raw-IPC metrics that do not rely on the choice of a reference machine. He proposes a new metric, the harmonic mean of IPCs. As a response to Michaud's paper, Eyerman and Eeckhout restate the case for WSU and HSU, arguing that raw-IPC metrics may give wrong conclusions [4].

3 Instantaneous throughput

Before modeling the average throughput of a specific TPEX, we discuss instantaneous throughput first. Instantaneous throughput is the throughput at a certain moment in time. Average throughput is a (weighted) average over time of the instantaneous throughput. Instantaneous throughput is the throughput of a specific coschedule, i.e., a specific job-to-core assignment. Because jobs come and go in a TPEX, the coschedule is not fixed, and the TPEX consists of a sequence of coschedules. The probability for a coschedule to occur depends on job arrival times, scheduling, job distribution and job sizes. These are determined by the TPEX assumptions.

3.1 General assumptions and definitions

The following assumptions and definitions are common for all the metrics considered in this study:

Assumption 1: The multicore is always fully utilized. We assume that there are always enough jobs available, so that the multicore is always fully used. This way, we calculate the maximum throughput, independent of job arrival times. As a result, for N jobs and K (not necessarily identical) cores, there are N^K possible coschedules.

Definition of a job. A job is determined by its *size* and *type*. A job's size is the number of instructions it executes. The job type represents the job's behavior, which we equate to that of a certain benchmark, or benchmark phase. The job inherits the IPS (average number of instructions executed per second)[2] of the benchmark (or benchmark phase) that defines its type, and it interferes with co-running jobs like the benchmark does. The job's size is independent from the benchmark's size. Different benchmarks define different job types. Distinct jobs can have the same type.

Assumption 2: A job has a homogeneous behavior. We assume that the IPS of a job is constant, i.e., its instantaneous IPS is equal to its average IPS.

Definition of throughput. Throughput is the amount of work done in a fixed time. Time is well defined and unambiguous, but the unit of work (UoW) is a matter of choice: instruction, job, iteration, request, etc.

Assumption 3: The UoW is either the instruction or the job. These two UoWs are general and can be applied to any program. In specific situations, the metrics in this paper can be adapted to model other UoWs.

In this paper, instruction throughput is expressed in instructions per second. Job throughput is defined as a relative number, normalized to the throughput of a reference machine. It quantifies the throughput increase over isolated execution on the reference machine. In this study, we consider single-threaded jobs, and the reference machine is a reference single core. Jobs of a given type have a fixed size, which is defined from the reference core. If the jobs' relative occurrence is

[2] If all cores have a constant and equal clock frequency, IPS can be replaced by the more common IPC (instructions per cycle).

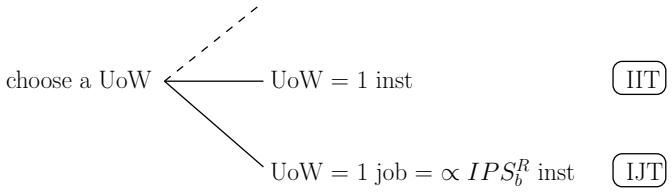


Figure 1: Taxonomy of instantaneous throughput metrics.

fixed, instruction throughput and job throughput are equivalent: instruction throughput equals job throughput times the average job size.

Assumption 4: For calculating job throughput, we fix the job size by assuming an equal sequential run time on the reference core for all jobs. This is a fair assumption in the absence of more knowledge on realistic workloads. Other job sizes can be used if more knowledge is available. Note that assuming an equal job size for all jobs (instead of an equal run time on the reference core) leads to a throughput number that is equivalent to instruction throughput, which is why we don't discuss job throughput when all jobs have equal sizes.

Notations. I_b is the size of a job of type b, $b \in [1, N]$, with N the *workload heterogeneity level*, i.e., the number of distinct job types encountered in a workload. $IPS_{c,s}^{MC}$ is the IPS of the job running on core c while executing coschedule s, with s an array of K elements giving the types of the jobs running on each of the K cores. We denote $s[c]$ the type of the job running on core c in coschedule s. IPS_b^R is the IPS of job type b on the reference core.

An instantaneous throughput metric is determined by the choice of a UoW, as shown in Figure 1. Here, the UoW is either the instruction or the job, where the job size is set proportional to IPS_b^R so that all jobs have an equal run time on the reference core. Other UoWs may be modeled, as suggested by the dashed line.

3.2 Instantaneous instruction throughput (IIT)

Instantaneous instruction throughput (IIT) is the number of instructions executed by the multicore per second. It is calculated as the sum of all IPSs of all jobs running concurrently in the coschedule:

$$IIT(s) = \sum_{c=1}^{K} IPS_{c,s}^{MC}. \tag{1}$$

3.3 Instantaneous job throughput (IJT)

Instantaneous job throughput (IJT) is the number of jobs executed by the multicore per second, the job size being proportional to IPS_b^R. The run time of job $b = s[c]$ running on core c while executing coschedule s is $T_{c,s}^{MC} = \frac{I_{s[c]}}{IPS_{c,s}^{MC}}$. Over a time T, job $s[c]$ has executed $\frac{T}{T_{c,s}^{MC}}$ times, leading to an IJT of

$$IJT(s) = \frac{\sum_{c=1}^{K} \frac{T}{T_{c,s}^{MC}}}{T} = \sum_{c=1}^{K} \frac{1}{T_{c,s}^{MC}} = \sum_{c=1}^{K} \frac{IPS_{c,s}^{MC}}{I_{s[c]}}.$$

I_b is proportional to IPS_b^R, so

$$IJT(s) = \sum_{c=1}^{K} \frac{IPS_{c,s}^{MC}}{IPS_{s[c]}^{R}}. \tag{2}$$

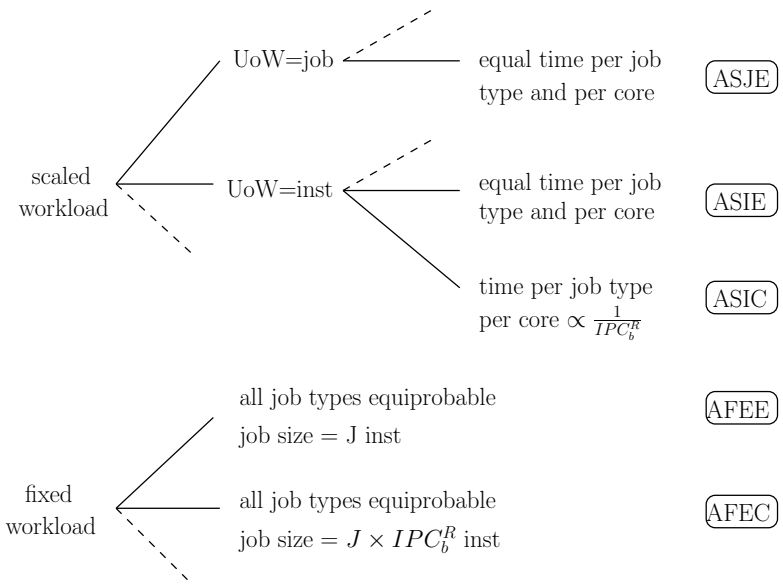


Figure 2: Taxonomy of average throughput metrics.

This formula is equal to the calculation of weighted speedup (WSU), aka STP [15, 2]. This derivation shows that WSU models the IJT of a coschedule, assuming all jobs have equal run times on the reference core.

4 Average throughput

Although instantaneous throughput shows how well certain job combinations co-run, what we want ultimately is the average throughput of a machine under a workload consisting of multiple jobs. The average throughput is the throughput of a theoretical TPEX. The TPEX should be close to the actual operation of the modeled machine and is independent of the simulation method (e.g., when and how jobs are started and finished). In particular, we would like to compute the average throughput of a TPEX just from isolated coschedule simulations, without actually simulating the full TPEX.

A TPEX is defined by its assumptions on job distribution, job scheduling and job sizes. These assumptions determine the probabilities for each coschedule to occur, from which average throughput can be calculated as a weighted average of instantaneous throughput. Alternatively, one can directly define the probabilities for each coschedule to occur, without expressing the TPEX assumptions.

In this section, we describe and elaborate some TPEXs and/or probability settings based on realistic assumptions. This sometimes leads to existing metrics, for which we have now a systematic meaning (i.e., the TPEX assumptions it models), and we also deduce new metrics. The list of metrics described here is not restrictive, new metrics can be deduced using different assumptions.

Naming of the metrics. We propose new names for the metrics we elaborate. These names are based on the corresponding TPEX assumptions, rather than on the formula that can be used to calculate them. For the average throughput metrics, the abbreviated names start with an 'A' for average, followed by the workload assumption: 'S' for scaled and 'F' for fixed. The

	Reference core IPC	Multicore IPC Co-run with A	Co-run with B
Benchmark A	2	1	0.5
Benchmark B	1	1	0.5

Table 1: Example experiment

two remaining letters describe the assumptions on the UoW and job type frequencies for scaled workloads, job type frequencies and job sizes for fixed workloads (see Figure 2, metric names are on the right side).

Definition of a workload. A workload is defined by the relative occurrence and the size of each job type on each core. We make two fundamentally different assumptions about the workload (see Figure 2): *scaled workload* and *fixed workload*.

With a scaled workload, we fix the run time of the jobs on the multicore. As a result, the number of instructions executed by a job and/or the frequency of a job type depends on the job's IPS on the multicore. Hence, the workload on two different multicore configurations might be different. Scaled workloads allow to obtain an exact formula for the average throughput.

With a fixed workload, the workload is the same on all machines. That is, the same jobs, with the exact same sizes and types, are submitted in the same order to all the machines being compared. Calculating average throughput in this case is harder, and often requires approximations. We discuss the two different workload assumptions in the next two sections.

Example experiment. To illustrate the different metrics, we use an example experiment. The workload consists of two job types (N=2). Jobs are executed on a symmetric dual core processor (two identical cores, K=2). Table 1 shows the IPC assumed for each job type. For each metric (Figures 3 to 7), we visualize the corresponding TPEX using these numbers. In the drawings, time is shown horizontally[3], and IPC vertically. Each box represents a job: the box width is the job run time, the height is the IPC, the box area is the job size.

4.1 Scaled workload

With a scaled workload, the total run time of each job type on each core is fixed, independent of IPS values. A fixed run time can be achieved by variable job sizes or by a variable occurrence frequency for each job type. We call this a scaled workload because the workload (job size and/or job type frequency) is scaled according to the performance of the different job types on a specific multicore. For example, a performance improvement for a specific job type leads to a larger size for that job, or more occurrences, possibly at the expense of other job types.

An example of scaled workload is an application that increases the quality of its output or offers better quality of service when executing on a machine with a greater throughput (for example, a video decoder that has to decode a frame in a fixed time period). Another form of workload scaling is when users scale the inputs of their jobs depending on the machines' performance, so that the job's run time is in an acceptable range. Gustafson's law for parallel programs is based on a similar assumption [5]. More generally, machines belonging to different generations are likely to execute different workloads.

Because the workload can differ between experiments, instruction throughput and job throughput are usually not equivalent. The next branching is therefore the selection of the UoW (upper tree in Figure 2).

[3]Each example displays a random job sequence that is compatible with the TPEX, but shows only a small time window. A full TPEX normally executes many jobs.

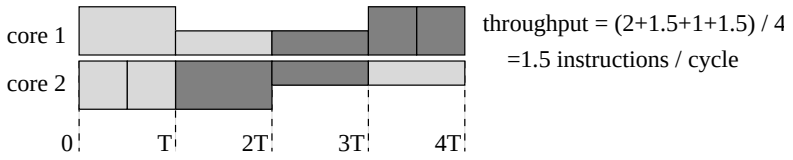


Figure 3: The ASIE metric. Example with 2 cores and 2 job types. The number of job instances and/or the job sizes are scaled so that, on each core, all job types use an equal time fraction.

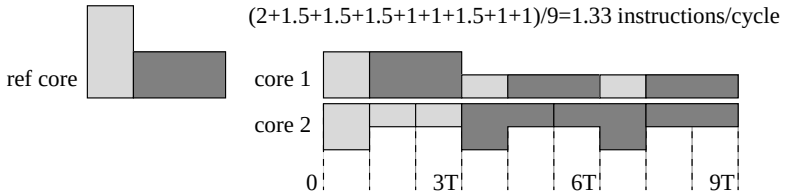


Figure 4: The ASIC metric. Example with 2 cores and 2 job types. Job durations are calibrated on the reference core, assuming equal size jobs. Job durations on the multicore are equal to the calibrated durations, all job types being equally likely on each core.

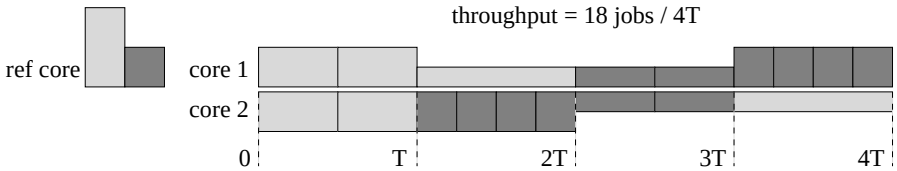


Figure 5: The ASJE metric. Example with 2 cores and 2 job types. Job sizes are calibrated to obtain equal durations on the reference core. On the multicore, the number of job instances is scaled so that, on each core, all job types use an equal time fraction.

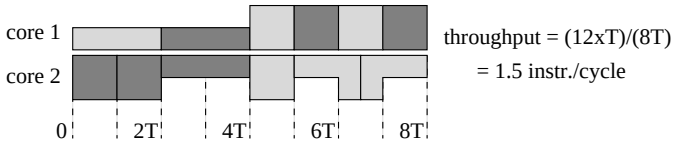


Figure 6: The AFEE metric. Example with 2 cores and 2 job types. All jobs have the same size. All job types are equiprobable.

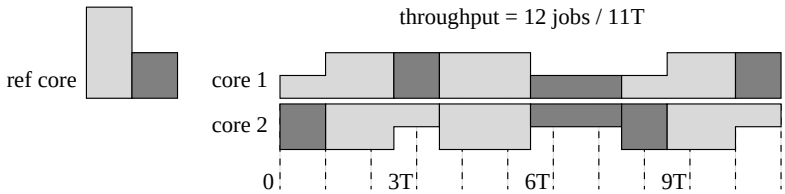


Figure 7: The AFEC metric. Example with 2 cores and 2 job types. Job sizes are calibrated to obtain equal durations on the reference core. On the multicore, all job types are equiprobable.

Instruction throughput. The easiest assumption to fix jobs' run times is to assume that all jobs have an equal run time on all cores. As a result, each coschedule is equiprobable. Therefore, the instruction throughput is the arithmetic mean of the IIT of all coschedules. The Average throughput for a Scaled workload using the Instruction as UoW, assuming Equal run times on each core (ASIE), is

$$ASIE = \frac{1}{N^K} \sum_{s=1}^{N^K} \sum_{c=1}^{K} IPS_{c,s}^{MC}. \tag{3}$$

Figure 3 shows the example TPEX.

Alternatively, it might be more realistic to assume that every job type has a different run time, but the run time of a specific job type is still fixed. For example, the run time of a job type equals its run time on a reference core, assuming equal instruction counts for all job types on the reference core. Jobs that run longer on the reference core (jobs with a low IPS) also take a proportionately longer time on the multicore. The probability for coschedule s to occur at a random time is the product of the probabilities of all the jobs in the coschedule:

$$Pr(s) = \prod_{c=1}^{K} \frac{T_{s[c]}^R}{\sum_{b=1}^{n} T_b^R} = \frac{1}{N^K} \prod_{c=1}^{K} \frac{\text{H-mean}(IPS_b^R)}{IPS_{s[c]}^R}, \tag{4}$$

with H-mean the harmonic mean over the N job types. The resulting instruction throughput is denoted ASIC (where 'C' stands for Calibrated run times):

$$ASIC = \sum_{s=1}^{N^K} Pr(s) \sum_{c=1}^{K} IPS_{c,s}^{MC}. \tag{5}$$

Figure 4 shows the example TPEX for ASIC.

Job throughput. For job throughput, we also initially assume that all job types have an equal total run time on each core. Because the job size is fixed for a given job type by assuming equal run times on the reference core, a fixed run time on the multicore is enforced by repeating each job type a different number of times. This again results in an equal probability for each coschedule to occur. The Average throughput for a Scaled workload using Job as the UoW assuming Equal run times on each core (ASJE) is:

$$ASJE = \frac{1}{N^K} \sum_{s=1}^{N^K} \sum_{c=1}^{K} \frac{IPS_{c,s}^{MC}}{IPS_{s[c]}^R}. \tag{6}$$

ASJE is the arithmetic mean of WSU on all coschedules. An example for ASJE is shown in Figure 5.

4.2 Fixed workload

A fixed workload means that every processor configuration executes the exact same workload. Lacking any knowledge on benchmarks' representativeness in real workloads, we simply assume that all job types are equally likely[4]. The corresponding TPEX is a multicore that is fed by an infinite FIFO queue that contains a random sequence of submitted jobs, using a simple FIFO scheduler: when a job is finished, the next job goes to the freed core, so that a core is never idle.

[4]With some knowledge on benchmarks' representativeness, we may want to give different weights to different job types.

Consequently, over a long time, we execute on each core approximately an equal number of jobs of each type. As a result, the coschedules that contain longer running jobs use more processor time. The probability of coschedules can also differ across machines, since the IPS and thus the run time of a job is determined by the multicore configuration.

With a fixed workload, the UoW does not matter. In particular, instruction throughput and job throughput are equivalent. We consider two different fixed workload metrics, as shown in Figure 2. Both metrics give the Average throughput for a Fixed workload assuming that all job types are Equiprobable. One metric assumes that all jobs have an Equal size (AFEE). The other metrics assumes Calibrated job sizes (AFEC), where the job size is proportional to IPS_b^R so that all jobs have an equal run time on the reference core. Figures 6 and 7 show example TPEXs for these two metrics.

Unlike for scaled workload metrics, there is no straightforward way to compute the coschedule probabilities. This is because the IPS of a job depends on the co-executing jobs, and one or several co-executing jobs may complete while that job is still executing. That makes it difficult to compute job run times, and we were not able to obtain a simple and exact analytical expression for the general case (any number of cores, any workload heterogeneity level, performance interference between co-executing jobs).

Exact analytical expressions exist for some special cases, in particular when jobs are "*insensitive*", i.e., when the IPS of a job does not depend on the co-executing jobs. We first consider this special case, and then we will discuss the general case of "*sensitive*" jobs whose IPS depends on coscheduled jobs.

Insensitive jobs. For insensitive jobs, the performance of a job of type b on core c, $IPS_{b,c}^{MC}$, is independent of the coschedule. Therefore the throughput of each core can be computed independently, the multicore throughput being the sum of the throughputs of all cores.

When all jobs have the same size (AFEE), the average instruction throughput of core c is the harmonic mean (H-mean) of $IPS_{b,c}^{MC}$ over all b:

$$\frac{\sum_{b=1}^{N} I_b}{\sum_{b=1}^{N} \frac{I_b}{IPS_{b,c}^{MC}}} = \frac{N}{\sum_{b=1}^{N} \frac{1}{IPS_{b,c}^{MC}}} = \text{H-mean}(IPS_{b,c}^{MC}).$$

The multicore throughput is:

$$\text{AFEEi} = \text{HIPS} = \sum_{c=1}^{K} \text{H-mean}(IPS_{b,c}^{MC}) \tag{7}$$

(where the 'i' after AFEE is a reminder that this formula only holds for insensitive jobs). In case of a homogeneous multicore, this is equivalent to computing the harmonic mean of all IPS values (HIPS), as proposed by Michaud [11]. Here we have generalized the HIPS for asymmetric multicores, showing that it corresponds to the AFEE assumptions in the case of insensitive jobs.

In case of calibrated job sizes (AFEC), the execution rate in jobs per second for a job of type b on core c is $\frac{IPS_{b,c}^{MC}}{IPS_b^R}$, hence the average job throughput of core c is

$$\frac{N}{\sum_{b=1}^{N} \frac{IPS_b^R}{IPS_{b,c}^{MC}}} = \text{H-mean}\left(\frac{IPS_{b,c}^{MC}}{IPS_b^R}\right).$$

The total job throughput is then the sum of the H-mean of speedups of each core:

$$\text{AFECi} = \text{HSU} = \sum_{c=1}^{K} \text{H-mean}\left(\frac{IPS_{b,c}^{MC}}{IPS_b^R}\right). \tag{8}$$

This is a generalization of the popular HSU introduced by Luo et al. [10]. HSU was proposed as an alternative for WSU, in order to account more for fairness between jobs. Here we show that it calculates job throughput for a fixed workload, assuming calibrated job sizes and insensitive jobs. The fact that we assume an equal occurrence of each job on each core intuitively explains that this metric incorporates a notion of fairness. Note that, in contrast to WSU which computes *instantaneous* job throughput, HSU computes the *average* throughput of a TPEX.

General case. In general, jobs are *not* insensitive, their IPS depends on the co-executing jobs. This is due to resource sharing (shared cache, shared memory bandwidth, shared SMT core resources, etc.). Finding an exact formula in the general case is very hard, but we can search for approximate formulas. The HIPS formula (7) and HSU formula (8) that we derived for insensitive jobs may be used as proxies for AFEE and AFEC respectively, replacing $IPS^{MC}_{b,c}$ with the harmonic mean of $IPS_{c,s}$ over all coschedules s such that $s[c] = b$. However, we show in Section 6 that HIPS is only a weak proxy for AFEE. Therefore, we searched for better approximations. Because of the space limit, we omit some details of the derivation. The UoW does not matter, and we conveniently assume that the UoW is the job. The average job throughput can be expressed as a weighted harmonic mean (WHM):

$$\text{WHM} = \frac{1}{\sum_{s=1}^{N^K} \frac{F_s}{\sum_{c=1}^{K} E_{c,s}}} \tag{9}$$

where F_s is the total work fraction executed by coschedule s and $E_{c,s}$ is the execution rate in jobs per second on core c under coschedule s.

Our approach is to compute F_s approximatively as if jobs were insensitive, a job of type b on core c having an execution rate of $E'_{b,c}$ jobs per second, independent of the co-executing jobs. This leads to the following approximation for F_s:

$$F_s \approx \left(\prod_{c=1}^{K} \frac{1}{E'_{s[c],c} \sum_{b=1}^{N} \frac{1}{E'_{b,c}}} \right) \times \frac{\sum_{c=1}^{K} E'_{s[c],c}}{\sum_{c=1}^{K} \frac{N}{\sum_{b=1}^{N} \frac{1}{E'_{b,c}}}} \tag{10}$$

The main difficulty is to define $E'_{b,c}$ in such a way that approximation (10) is reasonably close to the actual work fraction. The best approximation we found is based on the following definition of $E'_{b,c}$:

$$E'_{b,c} \triangleq \frac{\underset{\substack{s\in[1,N]^K \\ s[c]=b}}{\text{G-mean}} (E_{c,s}) \times \underset{\substack{s\in[1,N]^K \\ j\in[1,K],j\neq c}}{\text{G-mean}} (E_{j,s})}{\underset{\substack{s\in[1,N]^K, s[c]=b \\ j\in[1,K],j\neq c}}{\text{G-mean}} (E_{j,s})}) \tag{11}$$

where G-mean denotes the geometric mean.

For AFEE, all jobs have the same size, and any job size can be assumed. We can conveniently assume 1-instruction jobs, which means that the execution rate is:

$$E_{c,s} = IPS^{MC}_{c,s} \tag{12}$$

The AFEE-WHM approximation is obtained from (9), (12), (11), and (10).

For AFEC, the job size is proportional to $IPS^{R}_{s[c]}$. We can conveniently assume a job size of $IPS^{R}_{s[c]}$ instructions, which means that the execution rate is:

$$E_{c,s} = \frac{IPS^{MC}_{c,s}}{IPS^{R}_{s[c]}} \tag{13}$$

The AFEC-WHM approximation is obtained from (9), (13), (11), and (10).

5 Practical Considerations

The metrics elaborated in the previous section assume that all coschedules are simulated and $IPS_{c,s}^{MC}$ is known for every s and c. However, in practice, the number N^K of coschedules may be huge. Simulators are slow and it may be infeasible to simulate all N^K coschedules. We explain in this section how some metrics can be used with a relatively small sample of the coschedule space. For metrics that are not compatible with sampling, we show that symmetries may allow to reduce greatly the number of coschedules to simulate. But first, we argue that the workload heterogeneity N is likely to be small.

5.1 The workload heterogeneity level is generally small

The goal of an experimental setup is to be as close as possible to a realistic setup. Real multi-program workloads often consist of jobs with identical characteristics that are submitted to execution during a relatively short time frame, i.e., the workload is locally homogeneous. For heterogeneous workloads, the heterogeneity level (the number of distinct job types submitted during a time frame) is unlikely to be high. It would not be realistic to assume that the workload heterogeneity N is equal to the total number of benchmarks (or benchmark phases). Small values of N are more realistic. However, because we rarely know benchmarks' representativeness, we use several workloads formed from job types defined by our benchmarks. In general the workload space is huge and we must resort to sampling [24]. We obtain an average throughput for each workload, which is a *time* average for the TPEX associated with that workload. Eventually, we can aggregate the throughputs of all TPEXs with any averaging method, which can be viewed as a *spatial* average.

5.2 Sampling the coschedule space

If the number of cores K is not small, the number N^K of possible coschedules in a TPEX may be too large for being simulated completely. In this case, we may still obtain a meaningful proxy for an average throughput metric by sampling the coschedule space. However, not all metrics lend themselves to sampling. Moreover, different metrics may require different sampling methods. The average throughput metrics formulas considered in this study are defined on the full coschedule space. However metrics that compute an unweighted mean of per-coschedule terms lend themselves to random sampling [22, 24]. This is the case for ASJE, ASIE, HIPC and HSU. For these metrics, instead of computing the arithmetic mean or harmonic mean on the full population of coschedules, we can compute the mean on a random sample S, which provides an estimate of the average throughput. For instance the sampled ASIE and HSU formulas are

$$\text{ASIE-S} = \frac{1}{|S|}\sum_{s\in S}\sum_{c=1}^{K} IPS_{c,s}^{MC} \tag{14}$$

$$\text{HSU-S} = \sum_{c=1}^{K} \frac{|S|}{\sum_{s\in S} \frac{IPS_{s[c]}^{R}}{IPS_{c,s}^{MC}}} \tag{15}$$

The sample size $|S|$ is specific to a study [22, 24].

In weighted mean formulas such as ASIC, all coschedules are not equally important, and simple random sampling may not work. However if the weights are fixed and known, as is the case for ASIC, sampling is possible. In the ASIC formula, the weights of different coschedules may differ by orders of magnitude. Just taking a random sample and applying the ASIC formula

on it does not give a good proxy for ASIC. For ASIC, a sampling method that works well is to select coschedules with a probability proportional to their weights. The average throughput is then computed with an *unweighted* arithmetic mean, i.e., the sampled ASIE formula (14).

The AFEE-WHM and AFEC-WHM formulas are not functions of per-coschedule terms (the weight of a coschedule depends on other coschedules), hence sampling cannot be used. The AFEE-WHM and AFEC-WHM formulas should be used with the full population of coschedules.

5.3 Exploiting symmetries

Although the AFEE-WHM and AFEC-WHM formulas cannot be sampled, the number of simulations necessary to cover the full coschedule space may be greatly reduced if we can exploit symmetries. On a fully symmetric multicore, the mapping of jobs onto cores does not matter, all the mappings are equivalent. Different coschedules corresponding to the same *multiset* of job types yield the same IPS values after a permutation. The number of multisets of K jobs with N possible job types is $\left(\!\binom{N}{K}\!\right) = \binom{N+K-1}{K}$. There are generally much fewer multisets than coschedules. For example, with K=16 cores and N=3 job types, there are more than 40 million distinct coschedules but only 153 distinct multisets: 153 simulations are enough to obtain the IPS values for the N^K coschedules. Then, the average throughput, AFEE-WHM or AFEC-WHM, can be computed by mapping each of the N^K coschedule to the corresponding multiset[5].

When the multicore is not fully symmetric, we may still exploit symmetries. For instance, consider a multicore with P identical physical cores, each core having an SMT degree K/P. There are $\left(\!\binom{(\!\binom{N}{K/P}\!)}{P}\!\right)$ distinct configurations. For instance, with P=4 dual-SMT physical cores (K=8 virtual cores) and N=3 job types, 126 simulations are enough to cover the 6561 coschedules.

6 Analysis

When we choose a particular throughput metric to evaluate a microarchitecture, we are considering that the assumptions behind that metric are plausible. Hence it is logical to design the microarchitecture so that it behaves well under that metric. To make good design decisions, we need an intuition about the metrics we will use. For throughput metrics with simple formulas, such as ASIE, it is easy to understand how the individual IPS numbers impact the average throughput. However, metrics with complex formulas, like AFEE, are more difficult to understand. The goal of this section is to characterize throughput metrics and understand how they differ from each other. To this end, we define an abstract machine model.

6.1 Abstract machine model

Different metrics generally have different formulas. For some input numbers, these metrics give different conclusions. However in practice, IPS values are not random numbers, and different metrics may tend to give similar conclusions [23]. To compare different metrics, we use an abstract machine model, shown in Figure 8. This model mimics the IPS values produced by *symmetric*[6] multicores in an abstract way, without actually modeling the microarchitecture. It is based on an *interaction matrix* I_M that models interactions between job types. The coefficient in row b and column b' of the matrix represents the interaction between jobs of type b and b':

[5] Alternatively, the metric's formula can be modified to aggregate per-multiset values directly, taking into account the weight of each multiset (which is a multinomial coefficient).

[6] We did some experiments with an asymmetric version of the model but did not observe any qualitative difference with the symmetric case, so we present results only for a symmetric multicore model.

$$IPC_b^R = \text{IPCR} + (1 - \text{IPCR}) \times \frac{b-1}{N-1} \quad \text{for } b \in [1, N]$$

$$IPC_{c,s}^{MC} = IPC_{s[c]}^R \times \frac{1}{K-1} \sum_{\substack{j \in [1,K] \\ j \neq c}} I_M[s[c], s[j]]$$

$I_M[b, b']$ uniform random in $]1 - \text{SENSI}, 1]$, $b, b' \in [1, N]$

Figure 8: Abstract machine model for a symmetric multicore with K virtual cores ($K \geq 2$) and a workload heterogeneity level N ($N \geq 2$). Matrix I_M models interactions between job types. Parameter SENSI $\in [0, 1[$ models jobs sensitivity. Parameter IPCR models the dispersion of reference IPCs.

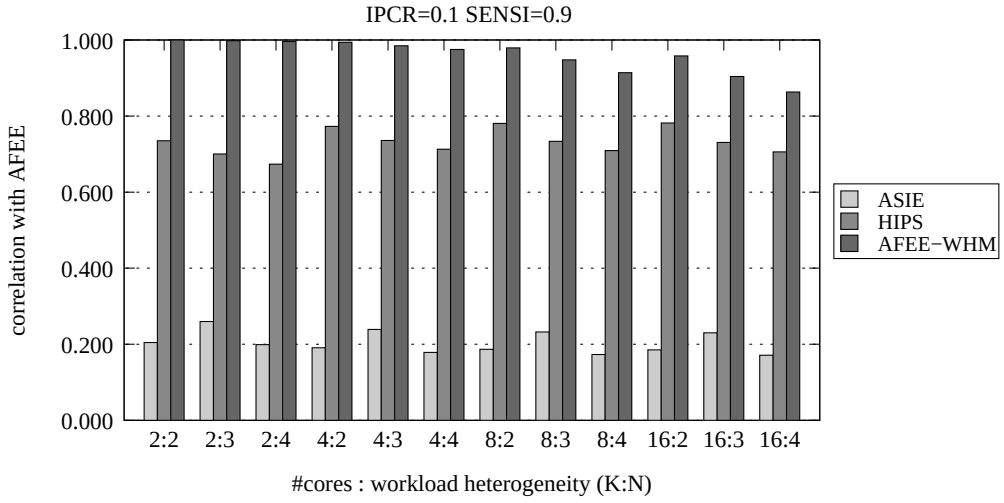


Figure 9: Correlation between AFEE and AFEE-WHM assuming IPCR=0.1, SENSI=0.9, for a number of cores $K = 2, 4, 8, 16$ and for a workload heterogeneity $N = 2, 3, 4$. Also shown for comparison are the correlations AFEE/HIPS and AFEE/ASIE.

the stronger the interaction, the lower the coefficient, and the more slowdown a job of type b experiences when co-running with a job of type b'. The model has 4 parameters: the number of cores K, the workload heterogeneity N, a parameter IPCR that quantifies the dispersion of reference IPC values, and a parameter SENSI that models job sensitivity. The reference IPCs of job types 1 to N follow an arithmetic progression, with $IPC_1^R = \text{IPCR}$ and $IPC_N^R = 1$. IPCR=1 means that all job types have the same reference IPC. SENSI=0 means that jobs are insensitive, and SENSI=0.9 means that jobs are quite sensitive.

To compare two throughput metrics/formulas $\mathcal{M}$ and $\mathcal{M}'$, we generate $P = 1000$ random interaction matrices according to the model of Figure 8. Each of these P matrices corresponds to a hypothetical microarchitecture. For each microarchitecture, we obtain two throughput numbers x_m and y_m corresponding to metrics $\mathcal{M}$ and $\mathcal{M}'$ respectively. Pearson's linear correlation coefficient ρ can be used to quantify the extent to which $\mathcal{M}$ and $\mathcal{M}'$ are equivalent [23]. When $\rho \in [0.9, 1]$, the two metrics are likely to give similar conclusions.

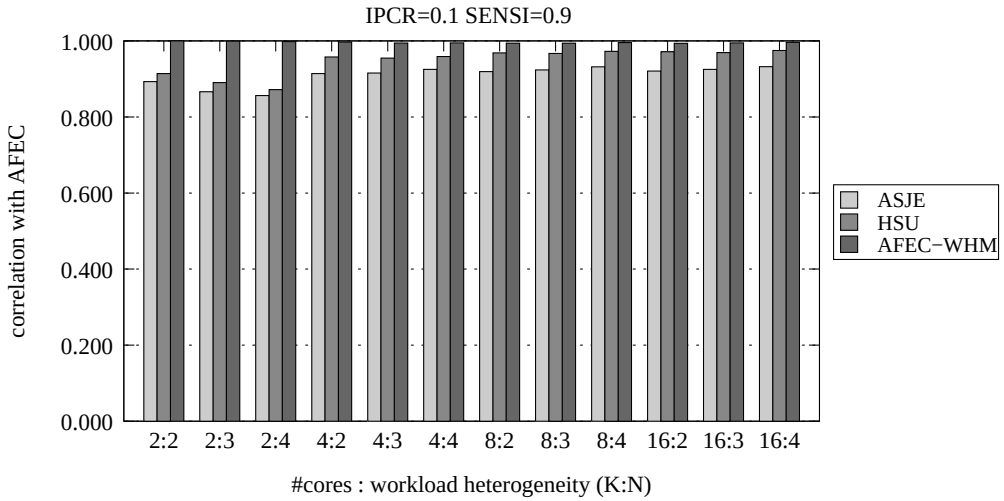


Figure 10: Correlation between AFEC and AFEC-WHM assuming IPCR=0.1, SENSI=0.9, for a number of cores $K = 2, 4, 8, 16$ and for a workload heterogeneity $N = 2, 3, 4$. Also shown for comparison are the correlations AFEC/HSU and AFEC/ASJE.

6.2 Approximating fixed workload metrics

We do not know exact analytical formulas for AFEE and AFEC. In this section, we evaluate how well approximate formulas approach these metrics. We do Monte Carlo simulations to obtain a reference value, using a method similar to the co-phase matrix of Van Biesbrouck et al. [21].

We show in Figures 9 and 10 that the AFEE-WHM and AFEC-WHM formulas are good proxies for AFEE and AFEC and may replace Monte Carlo simulations. The y-axis gives the linear correlation ρ. We vary the number of cores K, the workload heterogeneity N, and parameters IPCR and SENSI.

The correlation between AFEE and AFEE-WHM is greater than 0.9 except for K=16 and N=4. The correlation tends to decrease as we increase K and N. HIPS cannot be considered an excellent proxy for AFEE. AFEE-WHM should be used if one wants a proxy for the AFEE metric.

For the AFEC metric, the conclusion is different. Speedup values generally have less dispersion than IPC values. This is why AFEC-WHM is close to a perfect proxy for AFEC. However HSU and ASJE too are good proxies for AFEC. HSU is easier to understand than AFEC-WHM, and sampling can be used with it. Therefore the AFEC-WHM formula is not needed in practice, the HSU formula is sufficient.

6.3 Characterizing metrics

The abstract model of Figure 8 can be used to obtain some intuition about throughput metrics. Each microarchitecture is modeled with an $N \times N$ interaction matrix. An interesting question to ask is, which of the $N \times N$ interaction coefficients should we increase if we want maximum throughput gain?

To answer this question, we take a random interaction matrix I_M, generated as described in Figure 8. For a given metric, we evaluate the throughput corresponding to I_M. Then, we consider each coefficient of I_M separately, we multiply it by two, obtaining a new matrix I'_M

ASIE

1.01	1.01	1.01	1.01
1.05	1.05	1.05	1.05
1.08	1.08	1.08	1.08
1.11	1.11	1.11	1.12

ASJE

1.06	1.06	1.06	1.06
1.06	1.06	1.06	1.06
1.07	1.06	1.06	1.06
1.06	1.06	1.06	1.06

HSU

1.05	1.05	1.05	1.05
1.05	1.05	1.05	1.05
1.05	1.05	1.05	1.05
1.05	1.05	1.05	1.05

HIPS

1.14	1.14	1.14	1.14
1.03	1.03	1.03	1.03
1.02	1.02	1.02	1.02
1.01	1.01	1.01	1.01

AFEE-WHM

1.25	1.14	1.12	1.11
1.11	1.01	1.00	1.00
1.07	1.01	0.99	0.99
1.05	1.00	0.99	0.99

AFEC-WHM

1.04	1.05	1.05	1.05
1.06	1.04	1.06	1.05
1.06	1.05	1.04	1.05
1.05	1.05	1.05	1.04

ASIC

1.16	1.04	1.02	1.02
1.17	1.04	1.02	1.02
1.17	1.04	1.02	1.02
1.17	1.04	1.02	1.02

AFEE

1.33	1.13	1.08	1.06
1.06	1.02	1.02	1.01
1.03	1.01	1.01	1.01
1.02	1.01	1.01	1.00

AFEC

1.04	1.05	1.05	1.05
1.05	1.05	1.05	1.05
1.05	1.05	1.04	1.05
1.05	1.05	1.05	1.05

Table 2: Average speedup matrices for $K = 8$, $N = 4$, IPCR=0.1 and SENSI=0.9.

(differing from I_M in only one coefficient). We evaluate the throughput of I'_M, and we compute the throughput "speedup" of I'_M over I_M. This results in a *speedup matrix* for each metric. Entry b, b' in the speedup matrix corresponds to entry b, b' in the interaction matrix. We repeat this procedure 1000 times, every time with a random interaction matrix I_M, and report the average speedup matrix.

Table 2 show average speedup matrices for K=8, N=4, IPCR=0.1 and SENSI=0.9. Job type 1, the one with the lowest IPC_b^R, corresponds to the top row and left column in a matrix.

With the ASIE metric, the most important interactions are the ones in the bottom rows. To increase the ASIE throughput, we should try to accelerate high IPC jobs first.

If our goal is to increase the ASJE or AFEC throughputs, all the jobs are equally important, and we should focus our efforts on all the interactions. Notice that the HSU and AFEC-WHM formulas behave like the ASJE and AFEC metrics.

The AFEE metric has a unique behavior. It is interesting to contrast it with the HIPS formula. To increase the HIPS effectively, we should try to accelerate all low IPC jobs (here jobs of type 1). To increase the AFEE throughput, we should try to accelerate low IPC jobs *when they are co-running with other low IPC jobs*. The AFEE-WHM formula behaves approximately like the AFEE metric.

ASIC behaves like none of the other metrics. The interactions with the most impact on the ASIC throughput are in the leftmost column. That is, to increase ASIC, we should focus on coschedules containing some low IPC jobs and try to accelerate jobs in these coschedules, especially high IPC jobs.

6.4 Global comparison

Table 3 shows the linear correlation ρ for every pair of metrics/formulas, for K=8, N=3, SENSI=0.9 and IPCR=0.1. ASJE and AFEC/HSU/AFEC-WHM are almost equivalent. This can be explained as follows. The ASJE formula is a sum of per-core arithmetic mean of speedups. The HSU is the sum of per-core harmonic mean of speedups. For speedup values in the [0.5:1.5] range, we have H-mean $\approx$ A-mean $-$ Variance [11]. Because speedups have a relatively small variance, the harmonic mean is strongly correlated with the arithmetic mean.

	ASJE	HSU	HIPS	ASIC	AFEE-WHM	AFEC-WHM	AFEE	AFEC
ASIE	0.83	0.81	0.30	0.64	0.25	0.79	0.23	0.78
ASJE		0.95	0.75	0.74	0.61	0.94	0.56	0.92
HSU			0.74	0.71	0.60	0.98	0.55	0.97
HIPS				0.54	0.81	0.73	0.73	0.72
ASIC					0.79	0.70	0.70	0.69
AFEE-WHM						0.60	0.95	0.61
AFEC-WHM							0.56	0.99
AFEE								0.58

Table 3: Correlation for every pair of metrics/formulas, for K=8, N=3, SENSI=0.9 and IPCR=0.1

Table 3 also shows that ASIE is somewhat correlated with ASJE, hence with AFEC and HSU. This can be understood from the speedup matrices of ASIE and ASJE in Table 2. All the correlation coefficients are equally important in ASJE, whereas ASIE gives more weight to coefficients in the bottom rows. This is why ASIE and ASJE are not perfectly correlated. Nevertheless, except the topmost row, all the rows in the ASIE matrix have a non negligible impact. In that sense ASIE behaves similarly to ASJE[7].

As already shown in Figure 9 and Table 2, HIPS is a weak proxy for AFEE. The only advantage of HIPS over AFEE-WHM is that it can be sampled.

One of the main conclusions from table 3 is that the three popular throughput metrics / formulas, ASIE, ASJE and HSU tend to give similar conclusions. This is a property of our abstract machine model. To our knowledge, of the many studies that have used several throughput metrics, only few have their conclusion depend on which metric is considered. The model of Figure 8 reproduces this fact. Among the new metrics that we propose, AFEC is practically equivalent to the well known HSU for symmetric multicores, but ASIC and AFEE behave differently from the known metrics.

7 Real Simulation Data

The previous section compared the different metrics using synthetically generated data. This allowed to control the parameters and do a sensitivity analysis. This section contains results from a real simulation setup.

The goal of this study is to compare two power-equivalent processor configurations: a 4-wide out-of-order core with 2-way SMT (1 big core) compared to two 2-wide out-of-order cores (2 medium cores). We evaluate 12 2-job workloads, each consisting of 2 different SPEC CPU 2006 benchmarks. We checked that both configurations consume approximately the same amount of power using McPAT [9], and did performance simulations with Sniper [1].

This very small experimental setup allows to calculate the exact values for all metrics, without requiring approximations. Each workload requires 3 simulations for each configuration: two simulations with two identical copies for both benchmarks, and two simulation with the two different benchmarks. We evaluated all metrics: ASIE, ASIC, ASJE, AFEE and AFEC, as well as the HIPS and HSU approximations.

Figure 11 shows the throughput increase of 1 big core with SMT versus 2 medium cores for the 12 two-job workloads and the 7 metrics. All metrics make the same conclusion on what configuration is best for all workloads, even though this conclusion is different for different

[7] The correlation between ASIE and ASJE is lower when N=2.

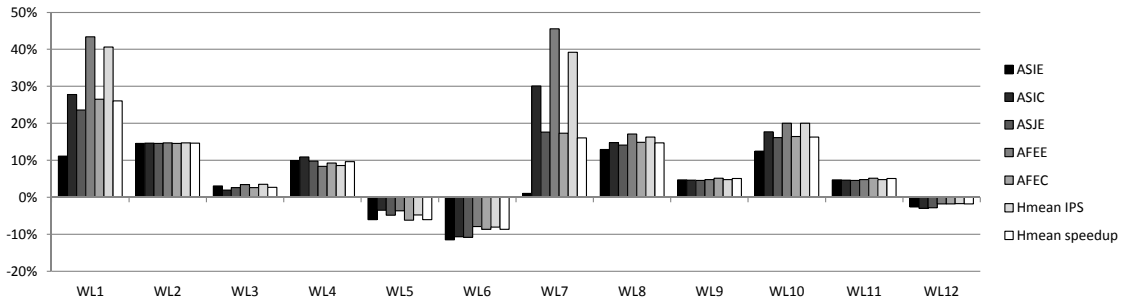


Figure 11: Relative throughput difference of 1 big core versus 2 medium cores for 12 workloads and 7 throughput metrics.

workloads. In some cases, all metrics even result in approximately the same relative throughput (e.g., workload 2). For these workloads, both jobs have a very similar behavior.

For other workloads, the different metrics give fairly different quantitative results. For workload 7, ASIE reports a throughput increase of 1%, while the throughput increase reported by AFEE is 46%. Using ASIE may lead to the conclusion that both configurations do not differ much, while AFEE suggests a substantial difference. In this workload, the two jobs show a very different behavior. One job is a high-IPC job whose IPC goes down on the big core, and the other job has a low IPC that increases on the big core. This confirms the finding that ASIE is more sensitive to high-IPC jobs, while AFEE rewards reducing interference between low-IPC jobs.

These results confirm some of the conclusions of the abstract model: ASJE, AFEC and HSU (3rd, 5th and 7th bar) are closely correlated and provide approximately the same quantitative conclusions. ASIE has only a weak correlation with these three. In this study, ASIE tends to give the lowest throughput increase for the big core, because high-IPC jobs usually have more interference on an SMT core than low-IPC jobs. ASIC, AFEE and HIPS, on the other hand, give more weight to the low-IPC jobs, leading to a larger speedup increase for these metrics. One noticeable difference with the abstract model is that for this small study, AFEE and HIPS seem to be more correlated than predicted by the abstract model (see workloads 1, 7, 8 and 10, 4th and 6th bar).

8 Limitations and Future Work

In this paper, we have elaborated some throughput metrics for multi-program workloads consisting of single-threaded programs. Here, we give hints and directions for future work.

8.1 Multiple multi-threaded programs

Workloads consisting of multiple independent multi-threaded programs become more and more relevant due to the growing number of multi-threaded programs and the interest in server consolidation. Throughput metrics similar to the ones described above can be deduced for a multi-program workload consisting of multi-threaded programs. The main difficulty is to define a meaningful UoW. The number of instructions in a multi-threaded program is usually not constant, so instructions and jobs defined using instruction count cannot be used as a UoW. An alternative for instructions could be to identify useful instructions, i.e., instructions that contribute to the actual work, excluding synchronization instructions.

8.2 Intelligent schedulers

All metrics in this paper assume a random scheduler. All jobs can run on all core types, independent of the co-running jobs. As a result, all coschedules have some probability to occur. An intelligent scheduler that tries to maximize throughput by scheduling jobs on core types based on their properties will have an impact on these probabilities and result in another average throughput number. For example, coschedules with a low instantaneous throughput will have a lower or zero probability to occur. Note that the instantaneous throughput metrics are still valid for intelligent schedulers, and the same taxonomy (e.g., scaled and fixed workload metrics) can be used to develop metrics assuming an intelligent scheduler.

8.3 Fairness

In this paper, we only discuss throughput, i.e., the amount of work done per unit of time. Another important concept in multi-program workloads is fairness: how is the throughput divided over the individual jobs? A situation where one job makes more progress than another job is considered less fair than a situation where all jobs make equal progress, even if the throughput in the first case is larger. Several specific fairness metrics have been proposed [2, 23]. However, the choice of the TPEX assumptions also has an impact on how fair the various jobs are treated in calculating average throughput. For scaled workload metrics, some jobs can be repeated more than others, which means that fairness is not taken into account. Fixed workload metrics, on the other hand, assume an equal occurrence for each job type, and therefore incorporate a notion of fairness in combination with modeling throughput.

9 Conclusions

There is no single throughput metric for multi-program workloads. Throughput depends on assumptions on job size, job distribution, scheduling, etc. A set of assumptions defines a throughput experiment, and the corresponding metric is the throughput of that experiment. Different assumptions lead to different metrics, which can have an impact on the conclusions of a study. Therefore, we should use metrics whose assumptions are plausible for a particular study.

We deduce multiple throughput metrics based on general and fair assumptions. In particular, we make a distinction between scaled workload metrics and fixed workload metrics. We identify the assumptions behind commonly used metrics, such as the IPC throughput, weighted speedup and the harmonic mean of speedups. We also propose new throughput metrics, whose calculation sometimes requires approximation. We explain how to use these metrics in a constrained simulation setup, and provide insight into how metrics are related and how to optimize for a certain metric, using both synthetic and real experimental data.

Our study can be used to select a meaningful metric for a multi-program study and can serve as a starting point to define a new metric for a specific setup. Understanding the assumptions behind a metric and its limitations leads to a better understanding of the actual results of an experiment.

References

[1] T. E. Carlson, W. Heirman, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *SC*, 2011.

[2] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, May 2008.

[3] S. Eyerman and L. Eeckhout. Probabilistic job symbiosis modeling for SMT processor scheduling. In *ASPLOS*, 2010.

[4] S. Eyerman and L. Eeckhout. Restating the case for weighted-IPC metrics to evaluate multiprogram workload performance. *IEEE Computer Architecture Letters*, April 2013.

[5] J. L. Gustafson. The consequences of fixed time performance measurement. In *Proc. of the 25th Hawaii Intl. Conf. on System Sciences*, 1992.

[6] A. Hilton, N. Eswaran, and A. Roth. FIESTA: a sample-balanced multi-program workload methodology. In *MoBS Workshop*, 2009.

[7] J. L. Kihm, T. Moseley, and D. A. Connors. A mathematical model for accurately balancing co-phase effects in simulated multithreaded programs. In *MoBS Workshop*, 2005.

[8] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *ISCA*, 2004.

[9] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*, 2009.

[10] K. Luo, J. Gummaraju, and M. Franklin. Balancing throughput and fairness in SMT processors. In *ISPASS*, 2001.

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

[12] H. H. Najaf-abadi and E. Rotenberg. The importance of accurate task arrival characterization in the design of processing cores. In *IISWC*, 2009.

[13] 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.

[14] Y. Sazeides and T. Juan. How to compare the performance of two SMT microarchitectures. In *ISPASS*, 2001.

[15] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading architecture. In *ASPLOS*, 2000.

[16] N. Tuck and D. M. Tullsen. Initial observations of the simultaneous multithreading Pentium 4 processor. In *PACT*, 2003.

[17] 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 *ISCA*, 1996.

[18] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *ISCA*, 1995.

[19] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Considering all starting points for simultaneous multithreading simulation. In *ISPASS*, 2006.

[20] M. Van Biesbrouck, L. Eeckhout, and B. Calder. Representative multiprogram workloads for multithreaded processor simulation. In *IISWC*, 2007.

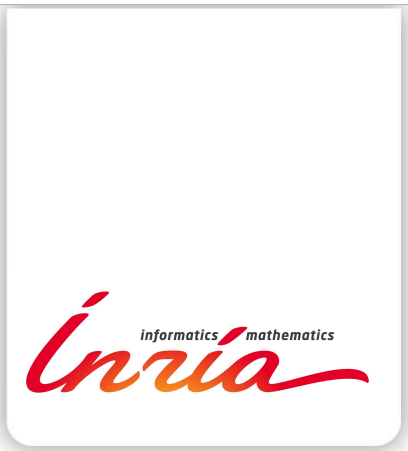
[21] M. Van Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *ISPASS*, 2004.

[22] K. Van Craeynest and L. Eeckhout. The multi-program performance model : Debunking current practice in multi-core simulation. In *IISWC*, 2011.

[23] H. Vandierendonck and A. Seznec. Fairness metrics for multi-threaded processors. *IEEE Computer Architecture Letters*, 10(1), January 2011.

[24] R. A. Velásquez, P. Michaud, and A. Seznec. Selecting benchmark combinations for the evaluation of multicore throughput. In *ISPASS*, 2013.

[25] J. Vera, F. Cazorla, A. Pajuelo, O. J. Santana, E. Fernández, and M. Valero. FAME: Fairly measuring multithreaded architectures. In *PACT*, 2007.



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