Sensible Energy Accounting with Abstract Metering for Multicore Systems

QIXIAO LIU, Universitat Politècnica de Catalunya and Barcelona Supercomputing Center
MIQUEL MORETO, Universitat Politècnica de Catalunya and Barcelona Supercomputing Center
JAUME ABELLA, Barcelona Supercomputing Center
FRANCISCO J. CAZORLA, Spanish National Research Council (IIIA-CSIC) and Barcelona Supercomputing Center
DANIEL A. JIMENEZ, Texas A&M University
MATEO VALERO, Universitat Politècnica de Catalunya and Barcelona Supercomputing Center

Chip multicore processors (CMPs) are the preferred processing platform across different domains such as data centers, real-time systems, and mobile devices. In all those domains, energy is arguably the most expensive resource in a computing system. Accurately quantifying energy usage in a multicore environment presents a challenge as well as an opportunity for optimization. Standard metering approaches are not capable of delivering consistent results with shared resources, since the same task with the same inputs may have different energy consumption based on the mix of co-running tasks. However, it is reasonable for data-center operators to charge on the basis of estimated energy usage rather than time since energy is more correlated with their actual cost.

This article introduces the concept of Sensible Energy Accounting (SEA). For a task running in a multicore system, SEA accurately estimates the energy the task would have consumed running in isolation with a given fraction of the CMP shared resources. We explain the potential benefits of SEA in different domains and describe two hardware techniques to implement it for a shared last-level cache and on-core resources in SMT processors. Moreover, with SEA, an energy-aware scheduler can find a highly efficient on-chip resource assignment, reducing by up to 39% the total processor energy for a 4-core system.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors); C.4 [**Performance of Systems**]: Measurement Techniques

General Terms: Design, Measurement, Performance

Additional Key Words and Phrases: Power modeling, energy accounting, resource allocation, modeling and estimation, chip multiprocessors, simultaneous multithreaded

This work has been partially supported by the Spanish Ministry of Science and Innovation under grant no. TIN2012-34557; the HiPEAC Network of Excellence, by the European Research Council under the European Union's 7th FP, ERC Grant Agreement no. 321253; and by a joint study agreement between IBM and BSC-CNS (no. W1361154). Qixiao Liu has also been funded by the Chinese Scholarship Council under grant no. 2010608015. Miquel Moreto and Jaume Abella have been partially supported by the Spanish Ministry of Economy and Competitiveness under Juan de la Cierva postdoctoral fellowship no. JCI-2012-15047 and Ramon y Cajal postdoctoral fellowship no. RYC-2013-14717, respectively.
Authors' addresses: Q. Liu, M. Moreto, J. Abella, F. J. Cazorla, and M. Valero, Barcelona Supercomputing Center, C/Jordi girona 29, Barcelona, Spain; emails: {qixiao.liu, miquel.moreto, jaume.abella, francisco.cazorla, mateo.valero}@bsc.es; D. A. Jimenez, Department of Computer Science and Engineering, Texas A&M University, TAMU 3112, College Station, TX 77843-3112; email: djimenez@cse.tamu.edu.

DOI: http://dx.doi.org/10.1145/2842616

ACM Reference Format:
Qixiao Liu, Miquel Moreto, Jaume Abella, Francisco J. Cazorla, Daniel A. Jimenez, and Mateo Valero. 2015. Sensible energy accounting with abstract metering for multicore systems. ACM Trans. Archit. Code Optim. 12, 4, Article 60 (December 2015), 26 pages.
DOI: http://dx.doi.org/10.1145/2842616

1. INTRODUCTION

Energy is becoming the most expensive resource in computing systems. This trend will continue as the price of energy continues to rise (increasing in recent years by up to 70% in several European countries [European Statistics 2014]). Under these circumstances, metering energy consumption of a computing system enables energy optimizations, ultimately helping to reduce system operation costs. In a data-center or supercomputing setting, charging users for energy rather than time makes sense because energy usage is more proportional to the cost of operations. The establishment of multicore and manycore as the *de facto* hardware paradigm across most computing domains, together with increasing core counts in each new generation, highlights the need for energy metering. Furthermore, applications are increasingly diverse, with many different providers and quite different energy profiles. Thus, accurate energy metering and optimization techniques are essential.

There are two main approaches when it comes to metering energy usage in a computing system:

—Per-Component Energy Metering (PCEM) derives the energy consumed by the main hardware components such as the CPU and memory. For instance, in the case of smartphones, several techniques [Carroll and Heiser 2010; Nokia 2012; Pathak et al. 2011] estimate overall system energy consumption, breaking it down per component (e.g., CPU, memory). Many proposals [Bircher and John 2012; McCullough et al. 2011; Pusukuri et al. 2009] use performance-monitoring counters (PMCs) or system events such as system calls to carry out such measurements. Power models rely on collecting data from a set of PMCs, and voltage and temperature information, to estimate power through correlation.
—Per-Task Energy Metering (PTEM) [Liu et al. 2013, 2014] estimates the energy actually consumed by each application simultaneously running in a multicore system. The main challenge of PTEM is dealing with shared hardware resources, as the energy consumption of applications significantly changes depending on the co-running applications. Unfortunately, PTEM metered energy for a given task is affected by the behavior of other tasks running on the same processor. We regard as inappropriate that the same program with the same inputs should be assigned different energy costs based on factors beyond the end user's control.

This article makes the case for Sensible Energy Accounting (SEA). Given a workload composed of n tasks[1] $T_1, T_2, \ldots, T_n$ running on a processor with n hardware threads (e.g., n single-threaded cores), SEA consists of estimating, for a given task T_i, the energy that it would have consumed if it had run in isolation with a given fraction of the hardware resources, denoted *fhr*. Thus, SEA does not give the actual energy consumption of a task, but rather an abstraction of the energy consumption that the end user can rely on to be fair and consistent.

Let us illustrate the concept of SEA and how it differs from PTEM with an example. We simulate several SPEC CPU 2006 benchmarks on a 4-core multicore architecture comprising a shared last-level cache (LLC)[2] and the PTEM technique [Liu et al. 2013].

[1]In this article, we use the term *task* to refer to hardware threads belonging to a single-threaded application.
[2]The experimental setup is described in Section 4.3.

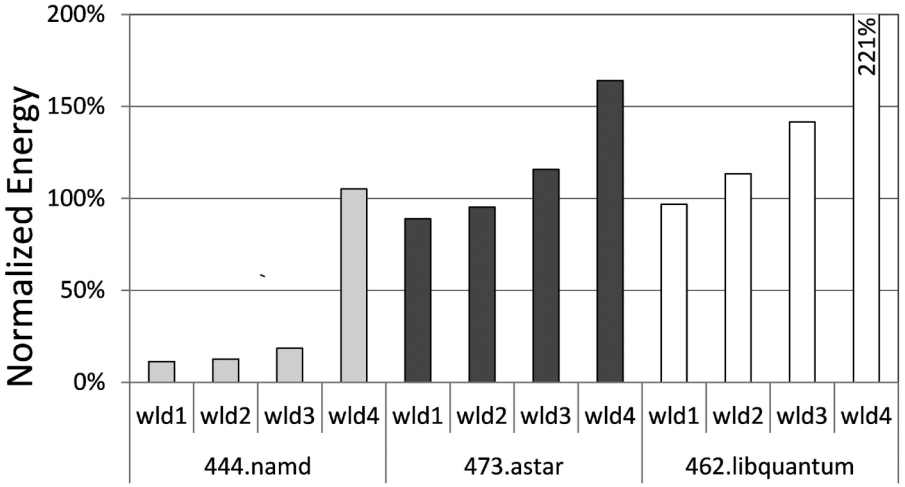

Fig. 1. Energy usage of *namd*, *astar*, and *libquantum* in different workloads with regard to their energy usage when executed in isolation with a fair share of resources.

We choose *namd*, *astar*, and *libquantum* benchmarks since they have different (LLC) utilization levels. We run each benchmark as part of 4 different 4-task workloads. The other 3 tasks in the workloads are only considered as co-runners, affecting the LLC behavior of the target benchmark. For instance, workload 1 comprises 3 copies of *namd*, which will cause almost no conflict to the target benchmark in the LLC. In contrast, workload 4 comprises 3 copies of *libquantum*, which makes the most intensive LLC use across those benchmarks. Workloads 2 and 3 have a mix of benchmarks to show some intermediate points in terms of LLC contention. Figure 1 shows the energy metered to the target benchmark in the workload, which is normalized to the energy the benchmark consumes when it runs in isolation with a fair share of the cache (i.e., 1/4 in our case). We observe that, despite the fact that each benchmark executes exactly the same instructions in each run, the energy it consumes significantly varies depending on the co-running applications. Sometimes the benchmark consumes much more energy, up to 2.2×, than when it runs in isolation with 1/4 of the cache, and other times it consumes as little as 11% of that.

This inconsistency is particularly problematic in environments in which users are charged for the usage of resources, including energy. Users running the same applications with the same inputs would observe different energy profiles for their applications, hence would unfairly receive different amounts billed. SEA helps by providing, for every task in a workload, the energy it would have consumed if run in isolation with a fair share of the shared resources. The energy charged is not exactly the energy consumed, but it is far more fair for end users (their billing solely depends on their own tasks) and still appropriate for the data-center operator since, typically, actual energy consumed is lower than energy measured due to using nonpartitioned shared resources. Note that those energy savings for the operator can be shared with end users by applying discounts for a mutual benefit. In this case, we assume that $fhr = 1/N$, where N is the number of hardware threads (cores in this case) in the system. The best value of *fhr* may vary across domains, as shown in the following sections.

In this article, we develop the concept of SEA from a theoretical point of view and discuss how it can contribute to different computing domains. Then, focusing on the on-chip resources, we present a low-overhead hardware mechanism to obtain SEA for a shared last-level cache in a multicore architecture. Our results show that SEA allows

savings of up to 39% of energy if used for scheduling purposes. Finally, we present an SEA mechanism for on-core resources taking into account simultaneous multithreading (SMT). Our results show that prediction error is only 5%, on average, for the core and between 4% and 8%, on average, for the whole chip when using SMT cores and a shared last-level cache. We also show how SEA attains much higher accuracy than other state-of-the-art mechanisms such as evenly splitting the energy across tasks or distributing it based on several metrics (number and type of instructions, and so on).

The rest of this article is organized as follows. Section 2 provides background on the different sources of energy consumption and existing approaches for energy metering and performance accounting. Section 3 explains our theoretical approach towards SEA. Section 4 presents SEA for a shared on-chip cache and its experimental results, while Section 5 presents the approach and evaluation for SEA for the core resources and integrates it with SEA for shared caches to cover the whole chip. Section 6 draws the main conclusions of this work.

2. BACKGROUND ON ENERGY METERING

SEA comprises two main building blocks: PTEM techniques and performance (CPU) accounting techniques. In this section, we elaborate on the state of the art for both.

2.1. Per-Task Energy Metering

As energy costs rise, interest in energy metering continues to increase in different computing domains, from data centers to smartphones [Carroll and Heiser 2010; Nokia 2012; Pathak et al. 2011]. PCEM techniques [Bircher and John 2012; McCullough et al. 2011; Pusukuri et al. 2009] focus on single-core architectures or multicores in which only one application is executed at one time and provide per-component energy estimations. However, processors incorporate an increasing numbers of cores, each implementing SMT, and running several applications with different energy profiles.

In this scenario, it is essential to determine energy consumption for each task. Shen et al. [2013] proposed a request-level OS mechanism to meter power consumption of each server request based on PMCs [Bellosa 2000]. The authors consider both active and maintenance power, attributing it to the responsible server requests. However, per-task energy estimates obtained with this approach cannot be validated since, as stated by the authors, *"Request executions in a concurrent, multi-stage server contain fine-grained activities with frequent context switches, and direct power measurements on such spatial and temporal granularities are not available in today's systems."*

Liu et al. [2013] covered this gap by proposing new hardware support for accurate PTEM in multicores. They propose tracking utilization of hardware resources for each task, including activities that they have incurred and the fraction of resources that they have used, to determine their fraction of energy used. Results show that, under different workloads, the variation of metered energy to some particular tasks can vary in the range of [−25%,40%] with respect to their average energy.

2.2. Performance Accounting in Multicores

The concept of SEA is inspired by CPU accounting [Luque et al. 2009] developed for multicores [Luque et al. 2012] and for SMT cores [Eyerman and Eeckhout 2009; Eyerman et al. 2006; Luque et al. 2013]. CPU accounting measures the CPU utilization of a given task during a period of time when it runs on a multithreaded processor. CPU utilization depends on both the time the task is scheduled on the CPU and the progress (or slowdown) the task experiences with the multicore. The latter is computed by determining which accesses to shared resources of a given task are delayed due to conflicts with other running tasks. For instance, if a task runs for a period of 1,000 cycles, in which it suffers a slowdown of 30%, its progress is 70% of what it would be with

Table I. PTEM and Performance Accounting in 2 Workloads

	h264ref	*calculix*	*povray*	*namd*
PTEM, EPI(nJ)	0.41	0.25	0.39	0.27
CPU utilization	68%	83%	75%	64%
	h264ref	*milc*	*sjeng*	*gcc*
PTEM, EPI(nJ)	0.73	0.70	0.43	0.82
CPU utilization	24%	86%	45%	75%

regard to its execution with a fair share of the resources. Thus, it is only accounted $1{,}000 \times 0.7 = 700$ cycles.

Performance accounting has been shown to be a powerful tool for performance optimization. For instance, it can be used to predict the performance with different degrees of contention to co-locate applications within the system. Results show that an individual application's performance can be improved by up to 22% and system utilization can be increased by 50% to 90% [Mars et al. 2010, 2011; Tang et al. 2011].

Using CPU accounting to scale energy estimated by PTEM as a way to achieve sensible energy accounting leads to inaccurate results. For instance, instruction mix and data locality have a large impact on energy that cannot be distinguished with CPU utilization. To illustrate this point, consider the execution of benchmark *h264ref* under two different 4-task workloads, as shown in Table I. In the first workload, *h264ref* incurs an Energy-Per-Instruction (EPI) of 0.41 nanojoules (nJ) and is accounted 68% of CPU utilization; in the second workload, *h264ref* incurs 0.73 nJ EPI and accounts for 24% of CPU utilization. One intuitive way to scale energy is to map CPU utilization to resource utilization. In this case, this method estimates that, under any resource utilization ru and EPI, *h264ref* would incur $SEA_{ru} = N_{ins} * ru * EPI$ (where N_{ins} stands for the instruction count). Thus, in the first workload, $SEA_{0.68} = N_{ins} * 0.279(0.41{*}0.68)$, and in the second, $SEA_{0.24} = N_{ins} * 0.175(0.73 * 0.24)$. As shown, the discrepancy across energy estimates in different workloads is huge across workloads (around 60%) if only CPU accounting is used; thus, SEA is needed.

2.3. Breaking Down Total Energy

Energy is conventionally divided into two main components: dynamic and leakage. In this article, we further divide dynamic energy into active and maintenance energy [Shen et al. 2013].

Dynamic active energy corresponds to the energy consumed performing those actions needed by the instructions executed, such as the energy used to read a register or to issue an instruction. Conversely, dynamic maintenance energy is the energy wasted in useless activities not triggered by any particular instruction, such as precharging bitlines in SRAM arrays when no one accesses those arrays, or the energy used by the selection logic in the issue queue when no instruction is ready. A perfect power gating scheme would avoid all maintenance energy consumption.

Finally, all energy wasted due to imperfections of the process technology (e.g., current leaks, short circuits from supply to ground, and so on) is considered leakage energy.

3. SENSIBLE ENERGY ACCOUNTING IN MULTICORES

In this section, we introduce our theoretical approach towards SEA showing some cross-domain applications of SEA and present the scenario considered in the rest of the article.

3.1. Theoretical Approach to SEA

SEA estimates an accounting for each task T_i while it runs with other tasks (i.e., as a part of a workload), the energy it would have consumed, $E_{T_i}^{fhr}$, if it had run in isolation with a certain fraction of hardware resources, *fhr*. Note that, in this abstract model, when running in isolation, T_i would be granted access to that fraction of resources, but is prevented from using more, although with shared resources T_i's usage may be more.

Interestingly, $E_{T_i}^{fhr}$ has to be estimated while T_i runs simultaneously with other tasks. In varied workloads, T_i can receive more or fewer resources than *fhr*, depending on co-runners. SEA must provide an accurate $E_{T_i}^{fhr}$, regardless of the particular usage of hardware resources that other tasks have[3].

Note that SEA's accounting model is conservative. It is possible that a given task may negatively affect co-running tasks by thrashing the cache, for example. In this case, SEA's abstract metering model would assign an overall energy cost to the tasks that is less than the actual cost to the provider. For this work, we assume that such situations would be dealt with by other means, for example, migrating cache-thrashing or other misbehaving tasks to cores where they can do less damage. SEA provides the means to detect those situations.

Problem Statement. Let's define $\mathcal{W}$ as a set of workloads composed of N tasks, in which a given task T_i is always present. Further define $W_j \in \mathcal{W}$ as $W_j = <T_i^{W_j}, T_{j_1}^{W_j}, \ldots, T_{j_{N-1}}^{W_j}>$, where $T_i^{W_j}$ corresponds to the actual execution of T_i in the workload W_j, and $T_{j_k}^{W_j}$ are any other tasks executing in the workload. In this scenario, the energy accounted to task T_i in a workload W_j, $E^{fhr}(T_i^{W_j})$, has to be as close as possible to the energy consumed in isolation with the same resource usage *fhr* by this task, $E_{T_i}^{fhr}$. This means that, with SEA, for any workload $W_j \in \mathcal{W}$, we expect that $E_{T_i}^{fhr} = E^{fhr}(T_i^{W_j})$.

Next, we illustrate two concrete applications of SEA: one particularly suitable for environments in which users are charged by the use of resources they incur and a second suitable across multiple domains.

Billing. When billing users for their use of resources, it is desirable to ensure that the same execution of the same application with the same input data result in the same charge. However, as shown in Figure 1, the energy consumed by a task can vary drastically depending on the co-runners. In this scenario, SEA can be deployed with $fhr = \frac{1}{N}$, where N is the number of hardware threads (i.e., the number of cores in a multicore processor) so that *fhr* corresponds to a fair share of the resources. Each task T_i is always charged $E_{T_i}^{1/N}$, which is independent of the actual energy consumed by the task, since the latter depends on T_i co-runners. If the actual energy consumed when running a workload E_{wld} is smaller than the energy accounted $\sum_{i=1}^{N} E_{T_i}^{\frac{1}{N}}$, the owner of the data center benefits from the $(\sum_{i=1}^{N} E_{T_i}^{\frac{1}{N}} - E_{wld})$ energy not actually consumed. This encourages the data-center owner to apply SEA, while the user enjoys workload-independent accounting. In our view, if $E_{wld} > \sum_{i=1}^{N} E_{T_i}^{\frac{1}{N}}$, it should be the data-center owner taking this extra cost, since assigning it to any task or proportionally to all tasks will break the principle of workload-independent energy accounting. As mentioned

[3]The SEA hardware support proposed in this article is able to estimate the energy a task should be accounted under several values of *fhr* at once, not just one. For the sake of clarity, we will be talking about a single *fhr* value without loss of generality.

Table II. Synthetic Example of Energy Consumption (in Arbitrary Units) under Different Fractions of Resources

	$E^{1/4}(T_i)$	$E^{2/4}(T_i)$	$E^{3/4}(T_i)$	$E^{4/4}(T_i)$
T_1	1.7	1.4	1.0	1.3
T_2	1.1	1.0	1.1	1.3

before, these situations can be prevented by properly allocating cache trashing tasks, for instance.

Energy optimization. Energy efficiency is pursued in all computing domains. Predicting the energy consumed by each task (or the system as a whole) under an arbitrary workload *a priori* is complex due to the many different ways the tasks composing the workload can interfere with each other. SEA can help in this respect. As we show later, SEA hardware support allows predicting the energy consumed by each task with an arbitrary fraction of the resources (*fhr*). For a discretized number of m valid values $\mathcal{F} = \{fhr_1, \ldots, fhr_m\}$ for *fhr*, SEA can predict the energy consumed by any task with any of those fractions of resources, resulting in m estimations. If this is done for every task in the workload, we can identify the resource partition that minimizes the total energy consumed by all tasks: $FHR_{min} = \min \sum_i E_{T_i}^{fhr_{i_j}}$ with $\sum_i fhr_{i_j} = 1$[4], and $i_j \in [1, N]$. Note that partitioning of shared resources is not needed by SEA. This example assumes it as a way to implement this optimization.

For instance, assume a 2-core processor with single-threaded (i.e., non-SMT) cores comprising a shared 4-way, last-level cache implementing way partitioning. Further assume two tasks T_1 and T_2 so that energy consumption under each different fraction of LLC is as shown in Table II. We can see that total energy is minimized when $FHR_{min} = <3/4, 1/4>$, as this leads to a total energy of 2.1 units. Any other partition leads to higher energy consumption. Also, if tasks are given the whole LLC space and executed serially, energy would also be higher (2.6 units) than for FHR_{min}.

3.2. SEA for On-Chip Resources in Multicores

SEA can be applied to any component of a computing system. In this article, we focus on on-chip resources in multicore processors, since the CPU is one of the major energy-consuming hardware blocks. In particular, we focus on a homogeneous multicore architecture deploying a shared last-level cache as the one described in Section 4.3.

SEA, as shown later, incurs some hardware overheads. As a result, SEA must be applied judiciously, taking into account the trade-off between accuracy in the energy predictions and hardware cost. With that goal, on the one hand, we apply SEA only to those resources that account for most of the energy consumed on-chip. We first consider the LLC of multicores. In a second step, we consider SMT cores whose resources are shared (i.e., the core itself, L1 data, and instruction caches). On the other hand, accounting for the energy for all possible fractions of resources would be infeasible. Hence, we focus on a set of predefined fractions. We consider each resource as a separate entity with a set of predefined granularities that represent the relative amount of resources assigned. In general, we will have granularities $g = \frac{M}{N}$, where $M \leq N$.

For the LLC, we consider only set-associative caches in this article, and define cache ways as the atomic granularity unit. For instance, in a 4-way LLC, N is 4, then, M is an integer in the range of $(0, 4]$. $\frac{1}{4}$ LLC for task T_i means that T_i can use 1 way in each set of the LLC. Note that, although SEA partitions the resources for accounting purposes, this is applied only to an abstract model to estimate energy consumption. SEA can target either shared or partitioned resources.

[4]Note that we could distribute less than 100% of the resources, but for the sake of simplicity, we assume that all resources are used by running tasks.

For the core, we use the fetch bandwidth as N, so that fetch bandwidth determines the partition granularity. Then, all other resources in the core, including all hardware blocks and bandwidths, are partitioned with the same degree. For instance, in an SMT core fetching up to 4 instructions per cycle, if T_i is given $\frac{1}{4}$ of the core, it receives $\frac{1}{4}$ fetch bandwidth, $\frac{1}{4}$ registers, $\frac{1}{4}$ issue queues entries, $\frac{1}{4}$ L1 ways, and so on. By doing so, we have a limited number of possible partitions for each hardware resource, and their granularities facilitate the hardware implementation of such partitions.

The main challenge for SEA is how to compute $E_{T_i}^{fhr}$ for any task and any valid fraction of the resources. In the next sections, we present our approaches in steps, first for a multicore processor in which only the LLC is shared, and then for a processor in which both core slices and LLC are shared. In both cases, we first propose an ideal SEA mechanism, then an efficient solution with hardware support that approximates such ideal values, assessing how our implementation of SEA performs in comparison with the ideal scenario.

4. SEA FOR MULTICORES: THE LLC

This section presents our approach for SEA in the presence of a shared LLC. First, we describe an ideal SEA model. Then, we propose an accurate, yet low-cost, implementation. Finally, we evaluate the accuracy of our implementation and illustrate the use of SEA for LLC in a practical case study.

4.1. Ideal SEA for the LLC

As explained in Section 2.3, dynamic active energy is proportional to the number of LLC accesses performed by T_i. Maintenance energy and leakage energy are proportional to the time and the fraction of the LLC used by T_i.

Sensible LLC active energy accounting. The key insight to accurately accounting for active energy, E_{act}, is that each action type in the cache incurs different energy consumption. For instance, a write operation requires more energy than a read. Hence, in the ideal case, we should collect the number of events of each action type that a task experiences with a given fraction $\frac{M}{N}$ of the LLC space, denoted $\frac{M}{N}LLC$:

$$E_{act}^{\frac{M}{N}LLC}(T_i) = \sum_{j=1}^{ActionTypes} Num_{action_j}^{\frac{M}{N}LLC}(T_i) \times E_{action_j}^{LLC}, \quad (1)$$

where $E_{action_j}^{LLC}$ stands for the energy per access to LLC of type $action_j$ (e.g., read-hit, write-miss, and so on). $Num_{action_j}^{\frac{M}{N}LLC}(T_i)$ is the number of LLC accesses of type $action_j$ performed by the task T_i if it is given M out of the N LLC ways.

The difficulty lies in estimating $Num_{action_j}^{\frac{M}{N}LLC}(T_i)$ for any valid value of M (number of cache ways) when T_i runs as part of a workload using a fully shared LLC. This is so because, under each workload, T_i may receive a variable amount of cache space, which affects the number of events of each action it has.

Sensible LLC maintenance energy accounting. The dynamic maintenance energy of the LLC is the energy consumed during idle periods due to useless activities such as clocking and precharging bitlines when no access occurs. Potentially, LLC maintenance energy consumption could be avoided if we turn off unused LLC parts (e.g., banks, lines, and so on). The fact that they are used by tasks prevents us from turning them off, so thus account maintenance energy proportionately to the cache space each task is entitled to use. Thus, maintenance energy to be accounted to T_i given a fraction $\frac{M}{N}$

of the LLC space is the same fraction of the total maintenance energy. Such total maintenance energy is the one that would be consumed assuming that the LLC is idle when T_i does not use it. Thus, maintenance energy is accounted as follows:

$$E_{main}^{\frac{M}{N}LLC}(T_i) = \frac{M}{N} \times P_{main}^{LLC} \times \left(ExecTime^{\frac{M}{N}LLC}(T_i) - \sum_{j=1}^{ActionTypes} Num_{action_j}^{\frac{M}{N}LLC}(T_i) \times Latency_{action_j}^{LLC} \right). \quad (2)$$

P_{main}^{LLC} is the LLC maintenance power, $ExecTime^{\frac{M}{N}LLC}(T_i)$ is the total time task T_i when executed with $\frac{M}{N}$ LLC ways, and $Latency_{action_j}^{LLC}$ stands for the latency of an action of type $action_j$. P_{main}^{LLC} and $Latency_{action_j}^{LLC}$ can be provided by the chip vendor. However, some parameters still need to be determined, such as $Num_{action_j}^{\frac{M}{N}LLC}(T_i)$, which is also needed to account active energy, and the execution time that would be had with exactly $\frac{M}{N}$ LLC ways, $ExecTime^{\frac{M}{N}LLC}(T_i)$. Note that such execution time cannot be easily estimated from the actual execution time when running as part of a workload sharing the LLC given that intertask interferences in the LLC may increase execution time, and T_i may use more than $\frac{M}{N}$ cache space, thus decreasing its execution time.

Sensible LLC leakage energy accounting. Finally, accounting leakage energy to T_i for a given fraction $\frac{M}{N}$ of the LLC space can be done based on the leakage energy per time unit, the fraction of cache space used, and the execution time of T_i, as follows:

$$E_{leak}^{LLC}(T_i) = \frac{M}{N} \times P_{leak}^{LLC} \times ExecTime^{\frac{M}{N}LLC}(T_i). \quad (3)$$

P_{leak}^{LLC} is the LLC leakage power. As for the maintenance energy, we need to determine $ExecTime^{\frac{M}{N}LLC}(T_i)$.

4.2. Implementation of SEA for the LLC

The accounting mechanism introduced in Section 4.1 is based on the estimation of the number of LLC accesses of each type (for active and maintenance energy accounting) and execution time task of T_i (for maintenance and leakage energy accounting) with $\frac{M}{N}$ ways of the LLC. Next, we describe affordable ways to approximate those values accurately.

Estimating access counts. Our approach to estimate the number of LLC accesses of each type when $\frac{M}{N}$ ways of the LLC are used relies on the Auxiliary Tag Directory (ATD) proposed by Qureshi and Patt [2006], which focuses on a least recently used (LRU) replacement policy. The LLC is shared among all tasks, each of which keep a local copy of the tag directory, the ATD, which is only updated with the accesses of the owner task. If the LLC implements LRU, one can predict whether an access would hit in the LLC for any number of cache ways M lower or equal to the actual number of LLC ways (N). This is so because LRU keeps in each set the position in the LRU stack of each address, thus the order in which they will be evicted if they are not reused. For instance, if in a 4-way LLC we access addresses A, B, C, D such that they are placed into the same set, the LRU stack, from the most recently used (MRU) entry to the LRU entry is as follows: $<D, C, B, A>$, thus meaning that if a new cache line is fetched into this set A will be evicted.

Based on the LRU stack, one can determine whether a given access would hit or miss with M ways (where $M \leq N$) by simply checking if it hits any of the M MRU entries. For instance, in our example, if we want to know whether accesses would hit in a 2-way cache given the LRU stack of the 4-way cache, we only need to check whether

it hits in the 2 most recently accessed entries. In our example, only accesses to D and C would be hits. In general, we can set up $N+1$ counters, $C_1, \ldots C_{N+1}$ so that C_i where $1 \leq i \leq N$ is incremented every time there is a hit in the way_i of any cache set, and C_{N+1} is incremented if X misses in all cache ways. Then, the number of hits and misses for $\frac{M}{N}$ ways of the LLC is obtained as:

$$Num_{hit}^{\frac{M}{N}LLC}(T_i) = \sum_{j=1}^{M} C_j \tag{4}$$

$$Num_{miss}^{\frac{M}{N}LLC}(T_i) = \sum_{j=M+1}^{N+1} C_j. \tag{5}$$

If different types of accesses have different energy consumptions (e.g., read and write operations), then $N+1$ counters need to be kept by each operation type so that each access updates the counter corresponding to its type. In practice, pseudo-LRU replacement is commonly used for LLCs. Although the ATD has been devised originally for LRU caches, it has been shown to be highly accurate if pseudo-LRU is used instead [Kedzierski et al. 2010]. Adapting the ATD to other replacement policies is left as future work and beyond the scope of this article.

Therefore, the ATD allows computation of the number of accesses of each type ($Num_{action_j}^{\frac{M}{N}LLC}(T_i)$). However, keeping one ATD per thread may be overly costly. Thus, Qureshi and Patt [2006] propose the Sampled ATD (SATD), which relies on keeping the tags only for a reduced number of the cache sets. For those sets, also computed is the overall hit probability for the different number of ways, $h_1, \ldots, h_N$, so that on access to a set not present in the SATD, which will likely be the case of most accesses, it can be predicted to be a hit or a miss. For that purpose, we use a *Monte Carlo* approach, which offers a high degree of accuracy and can be applied to each access at runtime. In particular, a random number RN is generated in the range $[0, 1]$. This RN and the actual hit probabilities for each number of ways, $h_1, \ldots, h_N$, are used to decide whether the current access should be a hit or a miss under each number of ways. Given that increasing the number of cache ways can only increase the hit rate[5], we have that $h_i \leq h_{i+1}$ for $1 \leq i < N$. In order to mimic a given hit probability h (e.g., $h = 0.7$), we use RN such that the access is a hit if $RN \leq h$ and a miss otherwise. Thus, we have to find the value of k where $1 \leq k \leq N+1$ so that $h_{k-1} < RN \leq h_k$. Such a k value indicates that the access is a hit for caches with $M \geq k$. For instance, in our example of a 4-way cache, we could have hit probabilities $0.2, 0.3, 0.7, 0.9$. If $RN = 0.6$, then $k = 3$ as RN is between h_2 and h_3, thus meaning that the access is assumed to be a hit if $M \geq 3$, thus if the thread is given 3 or 4 LLC ways. Similarly, if $RN = 0.95$, then $k = 5$, meaning that the access is a miss for any number of ways in the LLC.

The SATD trades hardware cost for accuracy: the lower the number of sets sampled, the lower the cost, but the lower the accuracy. The particular degree of sampling used for the SATD is indicated later in the Results section.

Estimating the execution time with a given cache fraction. CPU accounting for multicores, introduced in Luque et al. [2009], relies on using the ATD to decide whether each cache miss for a task T_i would hit or miss with a given fraction of the cache (typically a

[5]Given a cache with X ways, increasing its size by any number of ways (Y) so that its total number of ways becomes $X+Y$ can only have a hit rate higher or equal than with X ways only. This is so because the LRU stack for the X ways closer to the MRU position in the $X+Y$ cache is *identical* to the LRU stack of the X-way cache. Thus, all accesses hitting in the X-way cache will hit in the X ways closer to the MRU position in the $X+Y$-way cache. Then, the remaining Y ways may provide some more hits.

fair share of the cache space). A miss is caused by intertask interferences if the access hits in the task's local ATD and misses in the LLC. In that case, if the processor stalls, the cycles needed to serve the miss are not "accounted" to the task, meaning that the task would not suffer that miss, hence the associated penalty, if it had run a given share of the cache. Similarly, this CPU accounting mechanism accounts extra cycles to T_i in the case of an LLC hit that would have been a miss if T_i had run with a given fraction of the cache space.

This CPU accounting mechanism can be used to estimate the execution time that a task would have used to run with a given fraction of the resources, $ExecTime^{\frac{M}{N}LLC}(T_i)$. This helps in estimating the maintenance and leakage energy for a task since they are affected by the time that the task would run with a given fraction of the resources. Hence, we extend the CPU accounting mechanism for an N-way LLC to estimate the execution time of the task under any fraction of cache ways ($\frac{M}{N}$, where $1 \leq M \leq N$). CPU accounting uses the ATD as if the full cache is allocated to the task T_i. Cache accesses are considered to hit if they hit in the ATD, and to miss otherwise. In our case, we want to retrieve such information for different numbers of cache ways. The ATD provides such information by considering only those M entries closer to the MRU position. Thus, given a cache access, we can determine whether it would hit in any cache with $1 \leq M \leq N$ cache ways by checking the M ATD entries closer to the MRU position. Then, we can use such information to perform CPU accounting simultaneously for all different cache sizes. For each task, we need N cycle accounting (CA) registers, $CA_1, \ldots, CA_N$, which are updated as described in Luque et al. [2012], but where the decision on whether an access should be a hit or a miss – thus how CPU cycles need to be accounted – for CA_M is done assuming $\frac{M}{N}$ cache ways. Finally, note that CPU accounting can be implemented on top of the SATD with the same pros and cons as for counting the number of events of each type.

Overall, hardware requirements of the SEA for the LLC approach include an SATD for each task, the minimal logic and registers for accounting the CPU cycles per task introduced by Luque et al. [2012], and N+1 counters per task to obtain access counts for different numbers of LLC ways at once.

4.3. Evaluation

In this section, we assess the accuracy of SEA estimations for the LLC[6]. We also compare SEA with other intuitive methods that could be used to account LLC energy consumption. Finally, we illustrate by means of a case study how SEA can be used for energy optimization.

4.3.1. Experimental Setup. We use an enhanced version of SMTSim [Tullsen et al. 1998] extended with power models analogous to those of Wattch [Brooks et al. 2000] and McPAT [Li et al. 2009]. Those power models are built on top of the CACTI 6.5 simulation tool [Muralimanohar et al. 2009]. CACTI is a flexible tool modeling delay, energy (active and leakage) and area of cache memories and SRAM-based arrays. We assume CMP architectures with single-threaded cores (SMT cores are covered in Section 5). Each core has private data and instruction L1 caches, and a shared on-chip LLC accessed through a shared bus. Details can be found in Table III. We have 4 processor setups, 1-, 2-, 4-, and 8-core setups.

Benchmarks. We use traces collected from the whole SPEC CPU 2006 benchmark suite using the reference input set. Each trace contains 100 million instructions,

[6]Due to our proposed modifications to the microprocessor design, it is necessary to evaluate SEA in simulation rather than on real hardware.

Table III. Processor Configuration

Chip details	
Core count	2, 4, and 8
Core type	1-, 2-thread SMT (Section 5)
Core details	
Core type	out-of-order
Fetch, issue, commit bandwidth	4 instr/cycle
Issue queues size	48/48/48 entries for INT/FP/LS
Register file	80 INT, 80 FP
Inst L1	32KB, 4-way, 32B/line (2 cycle)
Data L1	32KB, 4-way, 32B/line (2 cycle)
Inst TLB	256 entries fully associative (1 cycle)
Data TLB	256 entries fully associative (1 cycle)
Shared L2 Cache	
Unified L2	2, 4MB, 16-way, 13/300 cycle hit/miss

selected using the SimPoint methodology [Sherwood et al. 2001]. Benchmarks in a workload are rerun until all of them have executed at least once.

Running all N-task combinations is infeasible, as the number of combinations is too high. Hence, we classify benchmarks into two groups depending on their memory behavior. Benchmarks in the memory group (denoted *MEM*) are those presenting an LLC miss rate higher than 1%, that is: *mcf*, *milc*, *lbm*, *libquantum*, *soplex*, *gcc*, *bwaves*, and *omnetpp*. The rest of the benchmarks are *CPU* (*ILP*) bounded and are denoted *ILP*. From these two groups, we generate 3 workload types denoted *I*, *M*, and *X* depending on whether all benchmarks belong to group *ILP*, *MEM*, or a combination of both.

We generate 8 workloads per group and processor setup. Benchmarks in each workload are randomly picked out from all the benchmarks of the corresponding type. In the case of *X*, half of the benchmarks belong to *ILP* and the other half to *MEM*. We do not put any constraint on whether benchmarks can repeat in a particular workload since the random selection of benchmarks is always performed out of the corresponding (original) group of benchmarks.

Metrics. In order to evaluate the accuracy of SEA, we use as a reference the actual energy consumption of a benchmark when it runs alone with the corresponding resource fraction. For instance, if we aim to estimate the LLC energy of a benchmark when it has only half of the LLC ways, the reference is a single-core processor setup with an LLC with half of the cache ways, for which the benchmark runs alone. Hence, in each experiment, we measure the *prediction error* of each model with respect to the actual energy consumed when one task runs with the specified fraction ($\frac{M}{N}$) of resources alone, which is computed as follows:

$$PredictionError = \left| 1 - \frac{EnergyAccount_{model}}{EnergyConsum_{\frac{M}{N}}} \right|. \tag{6}$$

4.3.2. Other Accounting Mechanisms. For the sake of completeness, we consider the energy estimates with some other intuitive and simplified models:

(1) *Evenly split model (ES)*. This model assumes that energy consumption is split evenly across tasks during the execution of the program. Note that this model is applicable only when $fhr = 1/N$.
(2) *Proportional to Access (PTA)*. PTA is a simple approach based on distributing the LLC energy proportionally to the number of accesses performed by each task.

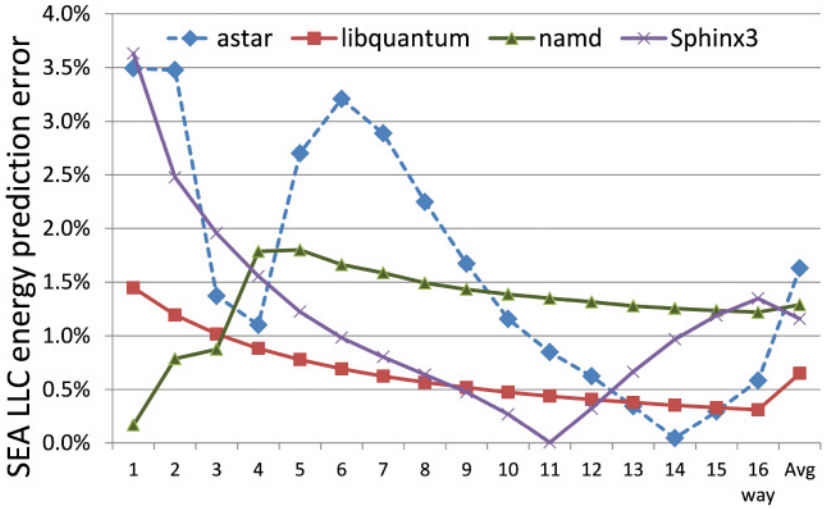

Fig. 2. SEA_{LLC} prediction error for a workload consisting of benchmarks *astar*, *libquantum*, *namd*, and *sphinx3* in a 16-way associative LLC.

(3) *PTEM*. As mentioned before, PTEM meters the energy consumption for each task based on the utilization of resources, including the activities incurred by each task and the fraction of resources used.

4.3.3. SEA_{LLC} Accuracy Evaluation. In our multicore architecture with single-threaded cores, the main sources of intertask interferences are the LLC and the shared bus. Our results show that the latter has negligible consumption in our architecture, thus we do not consider it for SEA as it does not pay off the extra hardware requirements.

We start analyzing SEA results for a given 4-task workload consisting of the following benchmarks: *namd* that has few LLC accesses regardless of the space available; *astar*, which accesses LLC often and whose LLC misses increase sharply when LLC space is decreased; *sphinx3*, which also has frequent accesses to LLC, but its LLC misses mildly increase when LLC space decreases; and *libquantum*, which has a large amount of LLC accesses but barely reuses the data in LLC, thus it is highly insensitive to the available LLC space and produces constant evictions.

From a single run of these benchmarks, SEA is able to obtain predictions of the energy that each benchmark would consume running in isolation under any partition of the cache. We evaluate SEA accuracy by comparing those predictions with the actual consumption each task has under each cache partition setup (see Figure 2). We can see that the error of SEA, which is computed as shown in Equation (6), is low for all cache partitions with a deviation of up to 4% and an average error always below 1.8%. In general, the prediction inaccuracy of SEA mainly comes from two sources: the estimation of the number of cache accesses by sampling the ATD and performance accounting based on estimating the number of extra cache misses with a given cache size and conflict misses incurred by co-runners. Some benchmarks show higher accuracy for a different cache partition. For instance, *namd* and *libquantum*, whose miss counts barely change with their varied given cache size, obtain highly accurate estimations across all cache sizes. Somewhat higher variations are observed for those benchmarks that are more sensitive to the space available, such as *astar* and *sphinx3*, with no particular trend with regard to the number of cache ways. Oscillations for different numbers of cache ways are mainly due to the fact that active, maintenance, and leakage energy are estimated separately, which may compensate or aggregate estimation errors depending

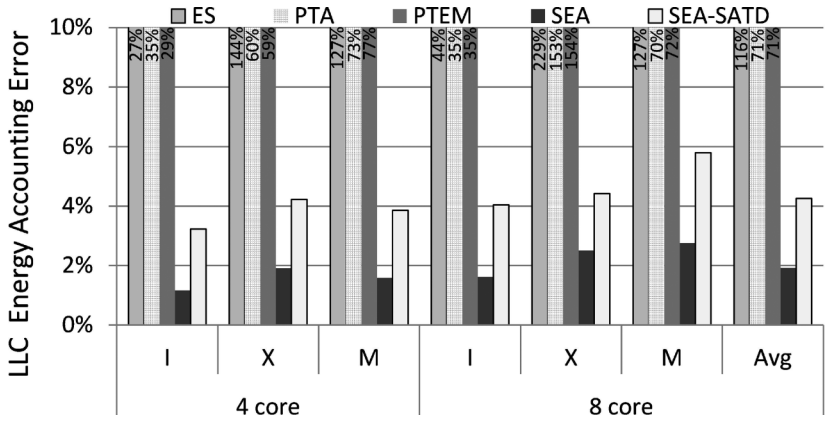

Fig. 3. LLC energy accounting error, under CMP 4- and 8-core setups, using I, X, and M type workloads.

Table IV. SEA-SATD Prediction Error Standard Deviation

	I	X	M		I	X	M
4 cores	3.5%	4.3%	3.7%	8 cores	4.8%	4.2%	6.1%

on whether each source of energy consumption is overestimated or underestimated for a given number of cache ways. Still, prediction error is rather low.

For the next experiment, we focus on the case in which $fhr = 1/N$, that is, SEA predicts when each benchmark receives a fair share of the LLC. Figure 3 shows the prediction error of the different models under 4-core and 8-core CMP setups: ES, PTA, and PTEM. Two versions of SEA are evaluated: with full ATD and with SATD.

As we can observe from the figure, *ES*, *PTA*, and *PTEM* fail to accurately predict the energy to account to each task. This is expected, as those models do not capture intertask interferences that impact energy consumed and how energy consumption for a task deviates from the reference. ES, PTA, and PTEM have prediction errors above 25% across all workload types and core counts and, on average, all of them produce deviations above 70%. On the other hand, SEA has consistent prediction accuracy, which has an error below 3% across all workload types and core counts, thus showing the excellent improvement of the method. When using SEA-SATD, whose hardware cost is lower, the error only grows to 4%. For the sake of completeness, Table IV shows the standard deviation for SEA-SATD. As shown, the variation of the prediction error across the whole set of workloads is moderate. Overall, SEA-SATD is highly accurate and far better than any state-of-the-art method.

4.3.4. Energy-Oriented LLC Allocation. In this section, we present a case study that shows how to use SEA as a powerful mechanism enabling energy savings. Similar approaches have been proven effective for performance optimizations [Mars et al. 2010, 2011; Tang et al. 2011]. Those approaches show that the performance gain could be significant when performance can be accurately accounted. By tracking the tasks running in a workload, SEA accurately estimates the energy consumed by each task under each number of allocated LLC ways, thus enabling efficient LLC space allocation algorithms with no need to run all programs under all configurations. In this section, we use a simplified scenario to show the potential for energy saving if we can choose the most

optimal resource allocation scheme for tasks in a multi-benchmark workload regardless of the system throughput and per-task performance. In this case, we assume a CMP architecture with a nonshared LLC, in which each task accesses its allocated LLC space exclusively. In this experiment, we have included the energy consumption of the memory. The memory system is simulated using DRAMsim2 [Rosenfeld et al. 2011], which is connected to our processor simulator. The power model in it is obtained from MICRON data sheets [Micron 2007]. Memory energy accounting is not in place and decisions regarding the most convenient cache partition are performed only based on core and LLC energy accounting. Thus, if memory energy accounting were in place, there would be potential for identifying better cache way partitions to further increase the energy saving. Sensible memory energy accounting would need a specific technique, which is part of our future work. Based on the fact that per-task memory energy metering has already been proposed [Liu et al. 2014] and SMT core and LLC energy accounting has been proved doable on top of energy metering, we do not expect any impediment in devising accurate memory energy accounting techniques.

At first, based on PTEM measurements, we can observe that benchmarks have various energy profiles with different numbers of allocated LLC ways. For some benchmarks, their consumed energy increases with more LLC ways. This is due to the correspondingly increased LLC power overlaps the reduction on execution time benefit from more LLC space. In contrast, the energy consumption of some benchmarks decreases with more allocated LLC ways. Analogously, this happens because their LLC misses reduce sharply with more cache space allocated, which significantly improves their performance. Also, there are several benchmarks with varying behavior. For those benchmarks, till a given point, allocating more LLC ways pays off because the energy saved due to the reduction in misses is higher than the extra energy consumed by those ways. Beyond that point, their LLC misses do not further significantly decrease and then, the energy consumed is increased.

Therefore, in this section, we classify benchmarks differently from what we showed in Section 4.3.1, since this helps to better understand the different characteristics across benchmarks. In particular, we divide programs into 3 categories: those whose energy increases as LLC space increases (i), those whose energy decreases as space increases (d), and the remaining ones that have a U-shape trend (u). i programs do not make efficient use of the cache space, thus increasing LLC space will simply increase their maintenance and leakage energy. They all have minimized energy consumption when only 1 LLC way is allocated. In contrast, d programs exploit LLC space efficiently, thus they minimize their energy consumed when they are allocated all LLC ways. Finally, u programs minimize their energy consumption with a number of ways larger than 1 and smaller than the whole cache space.

We compare the energy savings with the best LLC allocation with a fair share allocation in which each task gets the same number of cache ways. In Figure 4, bars show average energy saving across workloads in a particular category while the lines on top of them show the maximum savings. Workloads are built by combining half of the benchmarks of one type and half of another type.

As shown, the lowest average energy savings correspond to the cases in which all benchmarks are of type i (ii case) or of type u (uu case). This is expected, as i type benchmarks have a near-constant active energy consumption, and the optimal maintenance and leakage energy remain roughly constant regardless of how space is split. In the case of uu workloads, the baseline space distribution is already close to the optimal one as each program needs a fraction of cache space somehow in the central part of the distribution. In other cases, it is easy to find some benchmarks with different sensitivities to the amount of cache space, thus there are workloads with energy savings

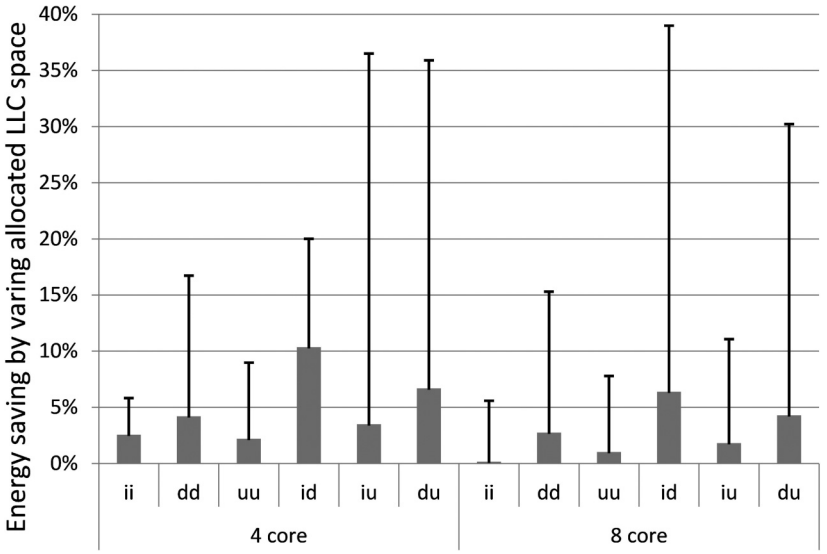

Fig. 4. Energy saving with varied LLC space allocation, comparing with fair allocation.

between 10% and 40%. These results confirm how SEA can be used to enable other energy-saving techniques.

5. SEA FOR MULTICORES: SMT CORES

This section introduces our approach for SEA in the presence of SMT cores. Following the same methodology in LLC, we first present the ideal SEA model in core and then a feasible, yet accurate, implementation. Finally, we evaluate the accuracy of our implementation for SMT cores and for a CMP architecture with shared LLC and SMT cores.

5.1. Ideal SEA for an SMT Core

Active, maintenance, and leakage energy are accounted separately, as in the case of the LLC.

Sensible SMT core active energy accounting. Active energy depends on the number of actions performed in each hardware component by a task T_i. Therefore, ideally, we would like to track the number of actions that would be performed by T_i in each resource if it was allowed to use $\frac{M}{N}$ of this resource exclusively. While defining $\frac{M}{N}$ of the resources is relatively easy for storage resources (e.g., caches, register files, issue queues, and so on), bandwidth resources (e.g., fetch bandwidth, issue bandwidth, and so on) can be split by allowing different tasks to use a fraction of the bandwidth [Huang et al. 2003b]. However, other resources, such as functional units, may need to be split in a different way. Given a partition granularity of N, if a task is allocated $\frac{M}{N}$ of the resources, this bandwidth splitting can be achieved exactly by allowing this task to use *all* resources during M out of N cycles. Still, in order to provide homogeneous behavior, we do so by providing the closest fraction to $\frac{M}{N}$ every cycle. For instance, if we have 4 adders and a task is allocated $\frac{1}{2}$ of the resources, it will get 2 adders every cycle. Similarly, if there are 2 adders and a task is allocated $\frac{1}{4}$ of the resources, it will get 1 adder every two cycles.

Active energy is, therefore, accounted as follows:

$$E_{act}^{\frac{M}{N}LLC}(T_i) = \sum_{k=1}^{Res} \left(\sum_{j=1}^{Actions(k)} Num_{action(k)_j}^{\frac{M}{N}k}(T_i) \times E(k)_{action_j} \right), \tag{7}$$

where Res stands for the number of different resources in the SMT cores, $Actions(k)$ for the number of action types in resource k, $Num_{action(k)_j}^{\frac{M}{N}k}(T_i)$ for the number of actions of type j performed by task T_i in resource k when given $\frac{M}{N}$ of this resource, and $E(k)_{action_j}$ for the energy of one action of type j in resource k.

Sensible SMT core maintenance energy accounting. In order to determine the maintenance energy to be accounted to one task T_i when given $\frac{M}{N}$ of the core resources, we use the same approach as in Liu et al. [2013]. First, we classify resources into two different categories: occupancy-based ($oRes$) and nonoccupancy-based ($nRes$). Maintenance energy for $oRes$ is accounted exactly as for the case of the LLC. Conversely, $nRes$ maintenance energy (e.g., selection logic in the issue queue when no instruction is ready) is simply split proportionally to the fraction of resources allocated. Thus, maintenance energy is accounted as following:

$$E_{main}^{\frac{M}{N}core}(T_i) = \sum_{k=1}^{oRes} E_{main}^{\frac{M}{N}k}(T_i) + \frac{M}{N} \times \sum_{k=1}^{nRes} \left(\sum_{x=1}^{ExecTime^{\frac{M}{N}core}(T_i)} E_{main}^{\frac{M}{N}k}(x) \right). \tag{8}$$

$E_{main}^{\frac{M}{N}k}(T_i)$ for $oRes$ is obtained as for the LLC (see Equation (2)). $ExecTime^{\frac{M}{N}core}(T_i)$ stands for the execution time of T_i when given $\frac{M}{N}$ of the core resources and $E_{main}^{\frac{M}{N}k}(x)$ is the maintenance energy consumed by resource k in cycle x when T_i executes with $\frac{M}{N}$ of the resources.

Sensible SMT core leakage energy accounting. Leakage energy can be accounted using the same methodology as in the LLC. Given a fraction $\frac{M}{N}$ of the core resources, leakage energy accounted to task T_i derives from the core leakage power per time unit (P_{leak}^{core}) and the execution time of T_i with $\frac{M}{N}$ of the core:

$$E_{leak}^{\frac{M}{N}core}(T_i) = \frac{M}{N} \times P_{leak}^{core} \times ExecTime^{\frac{M}{N}core}(T_i). \tag{9}$$

5.2. Implementation of SEA for an SMT Core

Tracking the activities of a given task T_i in all resources in the core is unaffordable. Instead, we propose periodically running a task T_i in isolation with a given fraction of the core resources and directly measure the energy, based on which we account the energy sensibly. Thus, we make use of the Micro Interval-Based Time Accounting (MIBTA) approach introduced in Luque et al. [2012], which has been used for performance accounting, and PTEM [Liu et al. 2013] for per-task energy measuring to derive the accounting energy to T_i. MIBTA divides execution time into time intervals in which the execution of running tasks are sampled alone in turn. During these sample phases, while one task has been granted the use of all resources in the core, the other running tasks are stalled temporarily. In our case, we need to carry out such sampling, but only allowing T_i to use $\frac{M}{N}$ of the core resources. The purpose of using these approaches is to sample T_i's energy consumption periodically when it uses $\frac{M}{N}$ of the core resources alone. During the sampling phases, PTEM can be used to measure

T_i's actual energy consumed in the core. PTEM provides accurate measurements of the active, maintenance, and leakage energy consumption in the core, thus their addition during the sampling intervals provides an accurate estimate of the energy accounting to T_i.

In the case of accounting active energy, the metered energy is nearly the energy that needs to be accounted. However, maintenance and leakage energy to account correspond to the fraction of maintenance and leakage energy of the whole core. Thus, SEA_{core} is estimated as follows:

$$E_{act}^{\frac{M}{N}core}(T_i) = P_{act,PTEM}^{\frac{M}{N}core}(T_i) \times ExecTime_{MIBTA}^{\frac{M}{N}core}(T_i) \tag{10}$$

$$E_{main}^{\frac{M}{N}core}(T_i) = \frac{M}{N} \times P_{main,PTEM}^{\frac{M}{N}core}(T_i) \times ExecTime_{MIBTA}^{\frac{M}{N}core}(T_i) \tag{11}$$

$$E_{leak}^{\frac{M}{N}core}(T_i) = \frac{M}{N} \times P_{leak,PTEM}^{\frac{M}{N}core}(T_i) \times ExecTime_{MIBTA}^{\frac{M}{N}core}(T_i). \tag{12}$$

$P_{act,PTEM}^{\frac{M}{N}core}(T_i)$, $P_{main,PTEM}^{\frac{M}{N}core}(T_i)$, and $P_{leak,PTEM}^{\frac{M}{N}core}(T_i)$ stand for the active, maintenance, and leakage power, respectively, estimated by the PTEM mechanism when running T_i during sampling periods. $ExecTime_{MIBTA}^{\frac{M}{N}core}(T_i)$ stands for the execution time predicted during the MIBTA phases when T_i is running with $\frac{M}{N}$ of the core resources.

Before entering the MIBTA phases (every 2.6 million cycles [Luque et al. 2013]), the execution of all tasks is stalled. Then, a controller restores the execution of a particular task to allow it to run alone in the core for 50,000 cycles to warm up. When time is up, the controller grants it another 50,000 cycles, during which some specified events are monitored to predict its execution time and energy consumed in this condition. The state of the other tasks is stored in the LLC when they get stalled, and their execution is restored after each MIBTA phase. In order to provide SEA_{core} capability, right after stalling the execution of the other tasks, the core is reconfigured to use $\frac{M}{N}$ resources. Adaptive processors (or reconfigurable processors) have already been studied to reduce power consumption [Albonesi et al. 2003; Dhodapkar and Smith 2002; Huang et al. 2003b]. In each component, such as the branch predictors and the buffers [Huang et al. 2003a]; register files [Abella and Gonzalez 2003; Homayoun et al. 2008]; issue queues [Cazorla et al. 2004; Folegnani and González 2001; Petoumenos et al. 2010]; caches [Albonesi et al. 2003; Qureshi and Patt 2006; Suh et al. 2002]; functional units; and fetch, decode, and issue bandwidth [Albonesi et al. 2003; Dhodapkar and Smith 2002; Huang et al. 2003b], power gating techniques have also been proposed with minimal area and energy overheads to power down different sections, with negligible impact on the delay.

With these techniques that we assume to be already in place, in the cache-like blocks, SEA_{core} can assign $\frac{M}{N}$ of the ways to T_i during the MIBTA phases with the remaining ways power gated. Similarly, during the sample phases, T_i is only allowed to use $\frac{M}{N}$ entries in the SRAM-like components, such as the issue queues and renaming registers. In contrast, nonoccupancy-based blocks are reconfigured in a way that $\frac{M}{N}$ of the bandwidth and the resources can be used in every cycle. If this fraction cannot be applied exactly, the closest value is enforced while still allowing T_i to progress. For instance, if T_i is entitled to use $\frac{1}{2}$ of the resources and there are 3 adders, it will be allowed to use either 1 or 2. In this case, we break the tie by providing the lowest value (1 adder) given that, for some resources, fractions can only be rounded up (e.g., if there is just 1 integer multiplier). SEA_{core} has considered ALUs; on-chip network bandwidth; as well as fetch, decode, issue, and commit bandwidth. Note that, during

Table V. SEA Hardware Requirements

	Description	HW overhead (8-core)
(S)ATD	ATD with sampled sets LRU stack distance counter	Total of 1920B per task, e.g., 0.7% of the LLC space
ITCA	logic to determine IT misses logic to account CPU cycles	Negligible
Reconfig. core	Branch predictor and buffers [Huang et al. 2003a], register file [Homayoun et al. 2008], issue queue [Cazorla et al. 2004; Folegnani and González 2001; Petoumenos et al. 2010], ALU, and fetch, decode and issue bandwidth [Albonesi et al. 2003].	Negligible
MIBTA	$CycleAccount_{MIBTA}$ $InstCommit_{MIBTA}$	2B per task 2B per task
PTEM	Energy Metering Registers Occupancy Counters	0.63% chip area overhead, 0.3% energy overhead [Liu et al. 2013]
SEA	**Energy Accounting Registers** **Target core and LLC resources**	**2 counters of 4B per task** **2 counters of 4B per task**

each MIBTA phase, some instructions may be squashed (i.e., when tasks are stalled to run one of them in isolation). They are reexecuted when the corresponding task is resumed since the program state (register contents) has been saved. In addition, the stalled task may have its used cache lines evicted by the running task, and thus incur extra cache misses. The result performance loss is detailed in Luque et al. [2013] and described in later sections.

5.3. Putting It All Together

We have introduced the SEA proposals in LLC and SMT core separately; the correlation must be taken into account when integrating them. In general, there is no conflict on the configurations of SEA_{LLC} and SEA_{core}, in the sense that one can use any fraction of their resources. Note that SEA_{core} needs to account energy of each task in the core sequentially by sampling them one after another in a particular order. However, the SEA_{LLC} does not impose any constraint on how tasks must run to account their energy. Therefore, while MIBTA, needed by SEA_{core}, samples one task at a time in any particular core, this can occur while other tasks run in other cores. Thus, the overhead of serializing task execution for sampling is limited by the degree of multithreading *in one core*, but not by the number of tasks in the whole processor chip. Therefore, one can sample tasks in different cores simultaneously in a way that scalability is not challenged when a large number of cores is in place.

Tasks interacting in the L1 cache have an impact on the number of LLC accesses, potentially causing inaccuracy in SEA_{chip}. To eliminate this effect, we monitor the number of LLC accesses per instruction during MIBTA phases when tasks run in isolation, thus have exclusive access to the L1 cache. The resulting LLC access frequency is assumed constant until the next MIBTA phase.

SEA hardware support and overhead. Regarding the hardware support-incurred overheads, SEA mostly inherits them from PTEM and MIBTA, as shown in Table V. Such overhead has been proved low, as can be seen in the same table with an 8-core configuration. Both PTEM and MIBTA require the SATD, whose area overhead is around 0.7% of the LLC [Luque et al. 2012, 2013; Qureshi and Patt 2006]. Few extra registers are needed by PTEM and MIBTA with negligible area overhead. In terms of energy, overheads are largely below 1%, which have been reported for PTEM, and they

have been shown not to grow with the number of cores [Liu et al. 2013]. MIBTA also introduces some performance overhead, which ranges between 1.0% and 3.2% [Luque et al. 2013]. Given that we have enhanced the MIBTA approach by allowing sampling tasks in all cores simultaneously instead of serializing task samplings across cores, the overhead is mildly reduced and does not grow with the number of cores. Our results show that MIBTA performance overhead remains around 2%, on average, regardless of the number of cores. In terms of energy, reconfiguring components in the core needs little extra logic to perform clock (or power) gating of unused parts during MIBTA monitoring periods. Such logic has been proven to have negligible area and power overhead and, in fact, it has been used to implement low-power mechanisms sharing the costs [Albonesi et al. 2003; Cazorla et al. 2004; Folegnani and González 2001; Homayoun et al. 2008; Huang et al. 2003a; Petoumenos et al. 2010]. Finally, SEA incurs very low overhead on its own due to those registers to store the accounted energy per task for the target core and LLC resources.

Other considerations. SEA may require considering temperature and voltage changes due to DVFS. We note that the LLC typically operates in a separate voltage domain, as its voltage cannot be easily decreased. Memory cells are sized to maximize integration, thus small transistors are used that are highly susceptible to process variations requiring high-voltage operation to read/write cells. Still, this is not a concern given that LLC active energy is low and idle banks are typically kept at lower voltages. Temperature variation is negligible in the LLC as its low activity keeps it at a mostly constant temperature.

Regarding the core, we note that DVFS becomes harder to use due to the need for decreased voltage for energy savings and increased minimum operating voltage to tolerate process variations [Bickford et al. 2008]. As a consequence, the acceptable voltage range narrows down in each technology generation. On the other hand, temperature variations in the core can occur. SEA can deal with voltage and temperature variations in both the core and the LLC by having as many energy constants (those that need to be provided by the chip vendor) as valid combinations of voltage and temperature ranges are allowed for the corresponding hardware block. For instance, if the processor can operate at 0.8V, 0.9V, and 1.0V, and temperature ranges are discretized as 320K–330K, 330K–340K and 340K–350K degrees, then 9 sets of constants are required to update the energy accounted to the tasks depending on the current voltage and temperature. Conversely, the ATD (or SATD) and the logic to predict whether accesses would hit in cache do not need to be changed given that such information is voltage and temperature independent. Overall, the overhead of this approach is low, as few hardwired constants need to be replicated.

Some Operating System (OS) support is needed to read energy accounting registers (EARs) when there is a context switch. This issue is analogous to the case of PTEM. In particular, we must expose to software the EARs for each hardware thread so that on a context switch the OS can reset it when a task is scheduled in and read it when it is switched out; its value is aggregated to the corresponding task. On a context switch, the contents of the ATD (or SATD) will likely differ from those that would be had if the task was run to completion without being scheduled out. This might have some impact on SEA accuracy. However, we have verified empirically that tasks typically fetch their working set to different cache levels in less than 200,000 cycles, which is less than 0.1ms in a processor operating at 2GHz. On the other hand, OS quanta vary from 4ms to 100ms for common Linux and Windows implementations, thus making context switch inaccuracy negligible; such inaccuracy falls below the inaccuracy of SEA method itself. Moreover, many tasks are not scheduled out on a context switch, further reducing such inaccuracy.

The actions performed by the OS working on behalf of a given task (e.g., on a system call) are assumed to be part of the task, thus the OS accounts such energy to that task. The energy accounted to other OS activities (i.e., "housekeeping" activities) can be evenly distributed across all running tasks, although any other policy can be followed to distribute OS energy based on the EAR registers exported by SEA.

With such OS support, applying SEA to multithreaded applications is simple since no additional hardware change is required. In fact, the OS can implement different mechanisms to account the energy to multithreaded applications by reading EARs and interpret the values in different ways. We illustrate some of these choices with a simple example: let us assume an N-thread, multithreaded application running on a N-core CMP, in which only the LLC is shared. In this case, we account each thread $E_{LLC}^{\frac{1}{N}}(t_i)$ as if the LLC is fairly shared across threads (cores) so that each one is given $\frac{1}{N}$ of the LLC. Upon the completion of one thread, the OS can choose to read the EAR of that thread and add its value to the total energy accounted to the application. Then, the OS can keep accounting the remaining threads in the same way until they all finish. Alternatively, the OS can read the EAR values of all active threads upon the completion of one thread, and add those values to the application's accounted energy. Then, the OS can account the remaining threads until another one finishes by assuming that they have extra LLC space to use. For instance, when the first thread finishes, each of the remaining threads will be accounted for $E_{LLC}^{\frac{1}{N-1}}(t_i)$ of the LLC space until another one finishes. The later approach is feasible as long as the thread completion and populating frequency do not exceed the OS quanta.

5.4. SEA_{core} Accuracy Evaluation

In this section, we evaluate the accuracy of the SEA approach in SMT cores. In order to account for the error of the core model, we discount the effect of the shared LLC in this experiment. In particular, the LLC energy accounted to a given task is obtained assuming that the full LLC space has been allocated to it. Therefore, energy variations can only come from the error of the core energy model.

We consider 2- and 4-way SMT core setups. Analogous to the LLC, the ES and PTEM models lack the flexibility and adequate accuracy to predict the energy one task has with a fraction of the core, thus we do not show them in the chart. On average, the ES model has an over 38% prediction error, while PTEM has an over 27% prediction error, when comparing their output with the energy one task should have consumed with the full core.

The prediction error for SEA is shown in Figure 5. We observe that, across all setups and types of workloads, SEA has stable prediction accuracy. For *X* type workloads, the average prediction error is rather higher than the others. We have also shown the standard deviation of SEA prediction error in the figure. While *X* type workloads also have higher variation than the others, the variation remains rather low for all workloads and setups. Nevertheless, SEA accuracy is still very high.

5.5. SEA_{chip} Accuracy Evaluation

In this section, we combine the SEA in the LLC and the core. Actually, SEA_{chip} is flexible with different combinations of SEA_{LLC} for $\frac{M}{N}$ of the LLC and SEA_{core} for $\frac{\hat{M}}{\hat{N}}$ of the core.

We analyze all configurations in which each task is accounted for half (*1/2 core*) or all (*1 core*) core resources, and for any number of cache ways between 1 and 16. Average off-estimation is shown in Figure 6 across the different configurations. The x-axis corresponds to the different number of cache ways (from 1 to 16). It can be seen

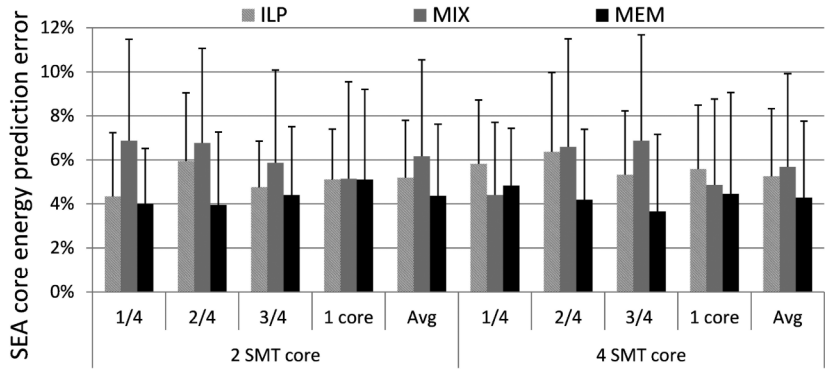

Fig. 5. SEA_{core} prediction error, under 2- and 4-SMT core setups, using I, X, and M type workloads.

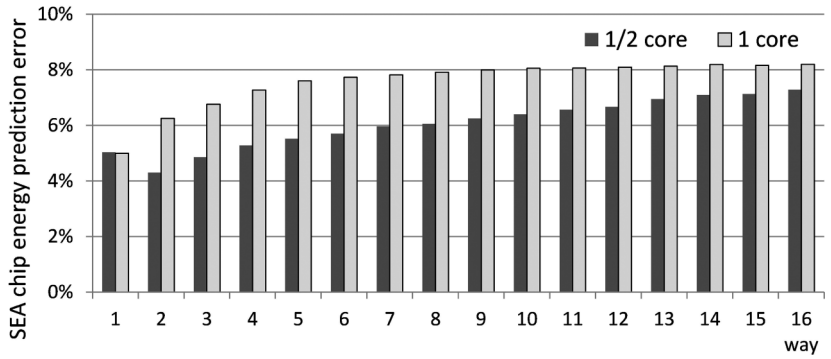

Fig. 6. SEA_{chip} prediction error for a 4-SMT core setup and 16-way LLC.

that the error is in the range of 4% to 8%, on average. In general, higher accuracy is attained when accounting energy for the *1/2 core* given that accuracy for the LLC is higher than for the SMT core, and the total energy to be accounted to the core under the *1/2 core* setup is lower. We also observe that higher accuracy is achieved for lower cache way counts. This occurs because miss rates are normally higher when fewer LLC ways are allocated, thus increasing the portion of active energy. Although the extra misses lead to more inaccuracies to the execution-time prediction, fewer LLC ways contribute low maintenance and leakage power, thus less impact, when compared with the increased but accurately estimated active energy.

Overall, SEA achieves very high accuracy estimating energy consumption under a given fraction of resources despite the fact that it is estimated under workloads in which many resources are shared in many different ways.

5.6. Energy Accounting Variability when Using ES, PTEM, and SEA

In order to illustrate the main conceptual differences between ES, PTEM, and SEA, in this section, we analyze the variation in terms of energy consumed and in terms of misprediction with regard to the energy that should be accounted. As for the actual energy, we make use of the ideal PTEM model proposed in Liu et al. [2013], which stands as an oracle version of PTEM that disregards the cost to measure energy. We consider that the per-task energy measured by this model is the best approximation of the actual energy consumed by tasks, thus, it is labeled as *ACTUAL* in the plot in Figure 7. Since all solutions compared (ES, PTEM, and SEA) have negligible energy

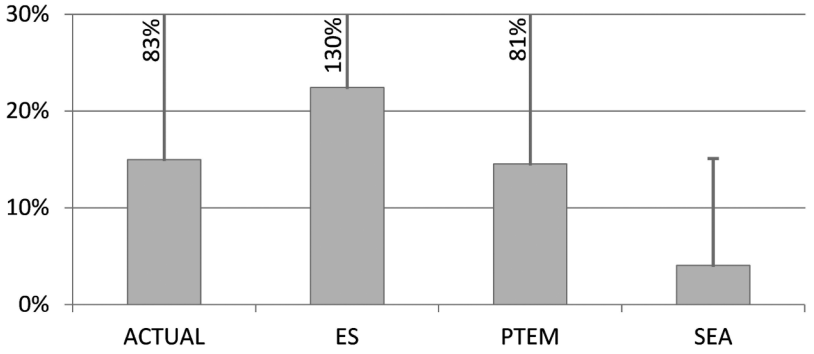

Fig. 7. The deviation of mispredicted energy account to tasks running in 8-task workloads under 4-core SMT setup and 16-way LLC.

impact in practice, the actual energy consumed is essentially the same, thus we just plot one column for *ACTUAL*. Note that accounting for a homogeneous share of the resources across tasks is the only case in which ACTUAL, ES, and PTEM can attain some degree of accuracy. In contrast, SEA is able to account energy for arbitrary fractions of the shared resources. Therefore, for comparison purposes here, we consider only a homogeneous share of the resources for each task.

In particular, we analyze the energy accounted to task T_i running in an SMT core of a 4-core, 16-way LLC, when half of the core resources and 2 ways of the LLC are accounted to it. In other words, T_i is accounted for exactly 1/8 of the resources of the processor, as it is able to run up to 8 tasks simultaneously. Figure 7 shows the average and maximum energy prediction errors. In particular, we obtain for each benchmark its range of variation (maximum minus minimum energy) with regard to its energy consumption when running alone with 1/8 resources; then, we report in the figure the average and maximum value across benchmarks.

We observe that the actual consumed energy has an average 15% prediction error across benchmarks and the maximum error reaches 83%. When using the ES model for energy accounting, we observe that variations are significant. On average, prediction error is 22%, while the maximum for one benchmark reaches 130%. This would mean that users would get 22% variations in the bills, on average, and those variations could reach 130% for the very same task. In the case of using PTEM, results of the actual implementation are very similar to those of the ideal PTEM model. On average, the prediction error is around 14% and in some cases it may be as high as 84%. This reflects the fact that many tasks may significantly overuse/underuse the resources with regard to a fair share of them. This affects their own energy consumption and co-runners' consumption. In contrast, SEA reduces the average error down to 4%, and maximum is 19% for one benchmark. These prediction errors are far lower than those of ES and PTEM, and can be hidden from end users to some extent by the fact that the cost per watt also varies along time. SEA is able to accurately predict the energy consumed with a fair share of the resources with negligible cost, as shown before, and allowing tasks to freely share resources.

In addition, when we account one workload with the energy accounted to $fhr = 1/8$ resources of all its tasks, comparing with its actual energy consumption, we found that the actual energy saved, on average, is 7.7% across all workloads because of resource sharing. Thus, on one hand, data-center operators can leverage the use of SEA to further reduce the actually consumed energy by finding an optimal point to co-locate tasks, as we show in Section 4.3.4. On the other hand, SEA can qualitatively apply the energy saving as a discount to end users as a mutual benefit.

6. CONCLUSIONS

The advent of CMPs allows running many tasks simultaneously, thus allowing resources to be shared and, generally, optimizing energy efficiency. Unfortunately, the energy consumed by a given task strongly depends on the set of co-runners, which create different intertask interferences. Therefore, energy consumption of a given task with a given set of inputs can change noticeably across different executions. If energy is used for billing, it is hard to defend charging end users largely different energy costs for the very same service.

This article introduces the concept of Sensible Energy Accounting (SEA). SEA allows accurate estimation of the energy that would be consumed by a given task if it were running with a given fraction of the resources, despite the fact that the task shares resources in a multitask workload. SEA thus opens the door to stable billing as well as energy optimizations in CMPs. Our results show that SEA provides highly accurate estimations for on-chip resources – as needed for billing – and can be used for scheduling purposes, achieving up to 39% energy savings.

REFERENCES

Jaume Abella and Antonio Gonzalez. 2003. On reducing register pressure and energy in multiple-banked register files. In *Proceedings of 21st International Conference on Computer Design*. 14–20.

David H. Albonesi, Rajeev Balasubramonian, Steven G. Dropsho, Sandhya Dwarkadas, Eby G. Friedman, Michael C. Huang, Volkan Kursun, Grigorios Magklis, Michael L. Scott, Greg Semeraro, Pradip Bose, Alper Buyuktosunoglu, Peter W. Cook, and Stanley E. Schuster. 2003. Dynamically tuning processor resources with adaptive processing. *Computer* 36, 12, 49–58.

Frank Bellosa. 2000. The benefits of event driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System(EW 9)*. 37–42.

Jeanne P. Bickford, Raymond Rosner, Erik Hedberg, Joseph W. Yoder, and Tomas S. Barnett. 2008. SRAM redundancy—Silicon area versus number of repairs trade-off. In *Advanced Semiconductor Manufacturing Conference*. 387–392.

W. Lloyd Bircher and Lizy K. John. 2012. Complete system power estimation using processor performance events. *IEEE Trans. Comput.* 61, 4 (2012), 563–577.

David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th International Symposium on Computer Architecture*. 83–94.

Aaron Carroll and Gernot Heiser. 2010. An analysis of power consumption in a smartphone. In *Proceedings of the USENIX Annual Technical Conference*. 21.

Francisco J. Cazorla, Alex Ramirez, Mateo Valero, and Enrique Fernandez. 2004. Dynamically controlled resource allocation in SMT processors. In *Proceedings of the 37th International Symposium on Microarchitecture*. 171–182.

European Statistics. 2014. Energy price statistics. http://ec.europa.eu/eurostat/statistics-explained/index.php/Energy_price_statistics.

Ashutosh S. Dhodapkar and James E. Smith. 2002. Managing multi-configuration hardware via dynamic working set analysis. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*. 233–244.

Stijn Eyerman and Lieven Eeckhout. 2009. Per-thread cycle accounting in SMT processors. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Vol. 44. 133–144.

Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. 2006. A performance counter architecture for computing accurate CPI components. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. 175–184.

Daniele Folegnani and Antonio González. 2001. Energy-effective issue logic. In *Proceedings of the International Symposium on Computer Architecture*. 230–239.

Houman Homayoun, Sudeep Pasricha, Mohammad Makhzan, and Alex Veidenbaum. 2008. Dynamic register file resizing and frequency scaling to improve embedded processor performance and energy-delay efficiency. In *Proceedings of the 45th ACM/IEEE Design Automation Conference*. 68–71.

Michael C. Huang, Daniel Chaver, Luis Pinuel, Manuel Prieto, and Francisco Tirado. 2003a. Customizing the branch predictor to reduce complexity and energy consumption. *IEEE Micro* 23, 5, 12–25.

Michael C. Huang, Jose Renau, and Josep Torrellas. 2003b. Positional adaptation of processors: Application to energy reduction. In *Proceedings of the 30th International Symposium on Computer Architecture*. 157–168.

Kamil Kedzierski, Miquel Moretó, Francisco J. Cazorla, and Mateo Valero. 2010. Adapting cache partitioning algorithms to pseudo-LRU replacement policies. In *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing*. 1–12.

Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *42th International Symposium on Microarchitecture.* 469–480.

Qixiao Liu, Miquel Moreto, Jaume Abella, Francisco J. Cazorla, and Mateo Valero. 2014. DReAM: Per-task DRAM energy metering in multicore systems. In *Euro-Par 2014 Parallel Processing*, *Lecture Notes in Computer Science*, Fernando Silva, Inês Dutra, and Vítor Santos Costa (Eds.), Vol. 8632. Springer, Berlin, 111–123.

Qixiao Liu, Miquel Moreto, Victor Jimenez, Jaume Abella, Francisco J. Cazorla, and Mateo Valero. 2013. Hardware support for accurate per-task energy metering in multicore systems. *ACM Transactions on Architecture and Code Optimizationb* 10, 4, Article 34, 27 pages.

Carlos Luque, Miquel Moreto, Francisco J. Cazorla, Roberto Gioiosa Alper Buyuktosunoglu, and Mateo Valero. 2009. CPU accounting in CMP processors. In *IEEE Comput. Archit. Lett.* 9, 2.

Carlos Luque, Miquel Moreto, Francisco J. Cazorla, Roberto Gioiosa, Alper Buyuktosunoglu, and Mateo Valero. 2012. CPU accounting for multicore processors. *IEEE Trans. Comput.* 161, 2.

Carlos Luque, Miquel Moreto, Francisco J. Cazorla, and Mateo Valero. 2013. Fair CPU time accounting in CMP&Plus;SMT processors. *ACM Trans. Archit. Code Optim.* 9, 4, Article 50, 25 pages.

Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. 248–259.

Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. 2010. Contention aware execution: Online contention detection and response. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*. 257–265.

John C. McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C. Snoeren, and Rajesh K. Gupta. 2011. Evaluating the effectiveness of model-based power characterization. In *Proceedings of the USENIX Annual Technical Conference*. 12–12.

Micron. 2007. Calculating memory system power for DDR3. *Micron Technical Notes* (2007).

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P. Jouppi. 2009. CACTI 6.0: A tool to understand large caches. *HP Tech Report HPL-2009-85*.

Nokia. 2012. Nokia Energy Profiler. http://nokia-energy-profiler.en.softonic.com/symbian.

Abhinav Pathak, Y. Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. 2011. Fine-grained power modeling for smartphones using system call tracing. In *EuroSys*. 153–168.

Pavlos Petoumenos, Georgia Psychou, Stefanos Kaxiras, Juan Manuel Cebrian Gonzalez, and Juan Luis Aragon. 2010. MLP-aware instruction queue resizing: The key to power-efficient performance. In *Architecture of Computing Systems - ARCS 2010*, *Lecture Notes in Computer Science*, Christian Meller-Schloer, Wolfgang Karl, and Sami Yehia (Eds.), Vol. 5974. Springer, Berlin, 113–125.

Kishore Kumar Pusukuri, David Vengerov, and Alexandra Fedorova. 2009. A methodology for developing simple and robust power models using performance monitoring events. In *Proceedings of the Annual Workshop on the Interaction between Operating Systems and Computer Architecture*.

Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th International Symposium on Microarchitecture.* 423–432.

Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. 2011. DRAMSim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters* 10, 1, 16–19.

Kai Shen, Arrvindh Shriraman, Sandhya Dwarkadas, Xiao Zhang, and Zhuan Chen. 2013. Power containers: An OS facility for fine-grained power and energy management on multicore servers. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, New York, NY, 65–76.

Tomothy Sherwood, Erez Perelman, and Brad Calder. 2001. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 3–14.

G. Edward Suh, Srinivas Devadas, and Larry Rudolph. 2002. A new memory monitoring scheme for memory-aware scheduling and partitioning. In *Proceedings of the IEEE Symposium on High Performance Computer Architecture*. 117–128.

Lingjia Tang, Jason Mars, Neil Vachharajani, Robert Hundt, and Mary Lou Soffa. 2011. The impact of memory subsystem resource sharing on datacenter applications. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*. 283–294.

Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy. 1998. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the International Symposium on Computer Architecture*. 533–544.

Received June 2015; revised October 2015; accepted October 2015