Hardware Support for Accurate Per-Task Energy Metering in Multicore Systems

QIXIAO LIU, Barcelona Supercomputing Center (BSC-CNS) and Universitat Politècnica de Catalunya (UPC)
MIQUEL MORETO, UPC and BSC-CNS
VICTOR JIMENEZ and JAUME ABELLA, BSC-CNS
FRANCISCO J. CAZORLA, Spanish National Research Council (IIIA-CSIC) and BSC-CNS
MATEO VALERO, UPC and BSC-CNS

Accurately determining the energy consumed by each task in a system will become of prominent importance in future multicore-based systems because it offers several benefits, including (i) better application energy/performance optimizations, (ii) improved energy-aware task scheduling, and (iii) energy-aware billing in data centers. Unfortunately, existing methods for energy metering in multicores fail to provide accurate energy estimates for each task when several tasks run simultaneously.

This article makes a case for accurate Per-Task Energy Metering (PTEM) based on tracking the resource utilization and occupancy of each task. Different hardware implementations with different trade-offs between energy prediction accuracy and hardware-implementation complexity are proposed. Our evaluation shows that the energy consumed in a multicore by each task can be accurately measured. For a 32-core, 2-way, simultaneous multithreaded core setup, PTEM reduces the average accuracy error from more than 12% when our hardware support is not used to less than 4% when it is used. The maximum observed error for any task in the workload we used reduces from 58% down to 9% when our hardware support is used.

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, per-task energy attribution, hardware counters, modeling and estimation, chip multiprocessors, simultaneous multithreaded

ACM Reference Format:
Liu, Q., Moreto, M., Jimenez, V., Abella, J., Cazorla, F. J., and Valero M. 2013. Hardware support for accurate per-task energy metering in multicore systems. ACM Trans. Architec. Code Optim. 10, 4, Article 34 (December 2013), 27 pages.
DOI: http://dx.doi.org/10.1145/2555289.2555291

New article, not an extension of a conference paper.
This work has been partially supported by the Spanish Ministry of Science and Innovation under grant TIN2012-34557 and the HiPEAC Network of Excellence. Q. Liu has been funded by the Chinese Scholarship Council under grant 2010608015.
Authors' addresses: Q. Liu, M. Moreto, V. Jimenez, J. Abella, F. J. Cazorla, and M. Valero, Computer Science, Barcelona Supercomputing Center (BSC-CNS), Barcelona, Spain.

DOI: http://dx.doi.org/10.1145/2555289.2555291

1. INTRODUCTION

Energy is one of the most—if not the most—expensive resources in a computingsystem. For instance, the electricity demand in data centers, where many computers work nonstop dissipating large amounts of power, shows the fastest growth among all sectors. In fact, current facilities consume several megawatts, enough to power small towns [Belady and Malone 2006]. Koomey [2011] estimates that worldwide energy consumption attributable to servers and data centers was more than 200 billion kilowatt hours (kWh) annually in 2010, and recent studies estimate the corresponding electrical cost to be $30 billion [Raghavendra et al. 2008]. Energy already accounts for 20% of the total cost of ownership in a large-scale computing facility [Hamilton 2009]. This cost doubles if we add the cost for the cooling infrastructure, implying that the total energy-related cost is already in the same order of magnitude as hardware-related cost (servers). Additionally, although server cost has remained almost constant over successive generations, energy cost is expected to rise [Barroso 2005].

Energy demand is also an issue for home computers. A typical desktop computer may use in the order of 100 to 200 Watts (W; the particular figure depends on the type of computer and peripherals), whereas laptops fit in a lower range (60–100W). The energy cost of running a computer can be computed as $\frac{Watts \times Hours\ of\ Use}{1{,}000} \times Cost\ per\ kilowatt\ hour$. Assuming that a computer runs for 20,000 hours during its lifetime (around 28 months nonstop) and a cost of 15 cents per kilowatt hour, the energy cost of a 150W desktop is $450. This figure already represents a significant fraction of the purchase cost of a computer. Moreover, as stated previously, energy cost will keep growing in the future [Barroso 2005].

It is our position that accurately measuring the energy consumed by each task[1] ina computer, instead of considering only the whole energy consumed by the computer, will have several applications. The benefits of such per-task energy metering are wide across different computing domains:

(1) *Selection of appropriate co-runners*. Task interaction in hardware shared resources may negatively affect tasks, hurting performance and increasing energy requirements. Per-task energy metering can help the OS scheduler or a runtime-based scheduler to decide which tasks must be run and when, thus reducing the total energy profile.
(2) *Energy/Performance optimization*. Although allocating more resources to a program may make it finish sooner, it could also increase its energy consumption. Thus, the net effect on total energy is not clear. Measuring the energy consumed per task would allow finding the processor setup (e.g., number of cores) and software setup (e.g., scheduling) that leads to the lowest system energy consumption.
(3) *Billing in data centers*. Data centers charge users for the use of their resources. The fact that costs will be dominated by energy makes billing systems more and more energy-centric; therefore, part of the bill directly depends on the energy consumed by users' running jobs. Measuring the energy that each task consumes, rather than evenly dividing the cost of energy among running tasks, would allow charging each user more fairly.

Unfortunately, current approaches to measure tasks' energy consumption evenly distribute computer system energy across all running tasks, as if all of them were using resources similarly. However, different applications may easily incur vastly different

[1]Throughout this article, we consider single-threaded tasks. In Section 9, we summarize how our approach also covers multithreaded tasks.

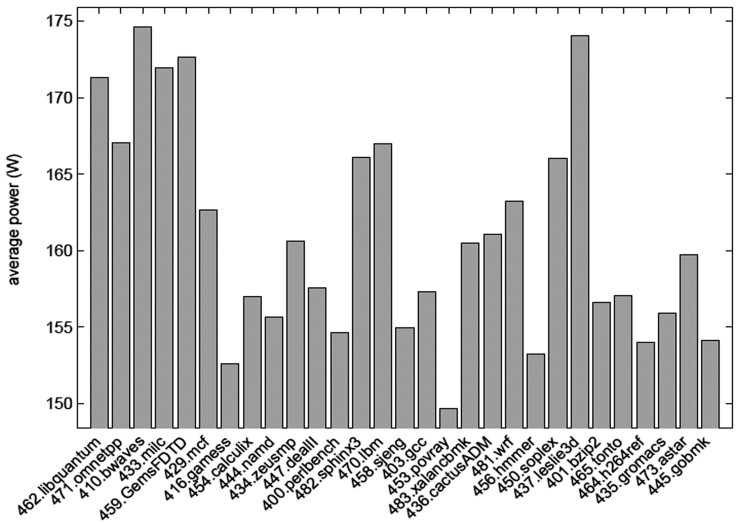

Fig. 1. Power consumption of SPEC CPU2006 benchmarks on a PS701 board with an IBM POWER7 processor.

resource utilization across similarly allocated resources. Such heterogeneous resource utilization translates into heterogeneous power dissipation per application; therefore, simply dividing power across running tasks is neither fair nor accurate enough.

To elaborate on the need for accurate per-task energy metering, Figure 1 shows the average power dissipation when executing all SPEC CPU2006 benchmarks on a POWER7-based system [Kalla et al. 2010]. As shown, different tasks incur different average power dissipation, with the maximum variation being 16%, between `453.povray` and `410.bwaves`. Hence, povray-like and bwaves-like workloads executing for the same amount of time incur significantly different energy consumption; however, the same amount of energy would be attributed to each program. Note that in this case workloads are fairly homogeneous given that they correspond to a single benchmark suite. More heterogeneous workloads including database processing, I/O-intensive applications, and high-performance ones exhibit higher power variations.

In this article, we make a case for accurate per-task energy metering. In particular, we propose an idealized reference approach to perform accurate PTEM based on the resource utilization of each task. We also present a simple, yet accurate, implementation of such an approach. We focus on the main shared hardware resources in current multicore processors: at chip level, we deal with the shared Last-Level Cache (LLC) and the network on chip; at core level, we consider simultaneous multithreaded (SMT) cores, which have a massive amount of shared hardware resources and represent the worst scenario for achieving accurate energy predictions with PTEM.

The benefits of PTEM extend to different computing domains, such as data centers, smartphones, or desktop systems. In this article, we take a cross-domain approach, in which instead of focusing on a given target environment, we analyze how to perform accurate per-task energy metering and what hardware/software support is required for an efficient implementation.

Overall, the main contributions of this article are as follows:

—We propose an accurate (yet idealized) approach to perform per-task energy metering based on per-task resource utilization. Our approach considers the utilization of

each hardware component in the chip (e.g., cores, caches) and its impact in dynamic, static, and leakage energy. Both single-threaded and SMT cores are considered by our approach. To the best of our knowledge, it is the first reference approach against which per-task energy measuring mechanisms can be compared.
—We show how state-of-the-art approaches such as those based on evenly distributing the energy consumed across running tasks fail to provide accurate enough per-task energy measurements.
—We propose efficient designs of our approach to perform per-task energy metering in multicore processors. We illustrate how our designs allow to accurately estimate the amount of energy that each task consumes in the chip by means of lightweight hardware mechanisms tracking activity and occupancy of the main resources in a per-task basis. In particular, we show how different trade-offs provide increasing accuracy at the expense of higher hardware and energy cost.

Our results over a variety of multicore processor setups and workloads, including SPEC CPU2006 and traces from a real High-Performance Computing (HPC) application called *wrf*, show that a low-cost implementation of our PTEM mechanism achieves tight per-task energy estimates with respect to an ideal nonimplementable model. For a 64-thread setup, 32 cores where each core is two-way SMT, PTEM reduces the average accuracy error from more than 12% when evenly splitting energy over running tasks to less than 4% when our low-cost hardware support is used. The maximum observed error for any task in the workload we used is reduced from 58% to 9% when our hardware support is used.

The rest of this article is organized as follows. Section 2 provides background on energy consumption and existing approaches for energy metering. Section 3 presents our idealized approach to perform per-task energy metering. Its efficient hardware implementation is described in Section 4. The experimental setup, intracluster results, and full processor results are detailed in Sections 5, 6, and 7, respectively. Next, in a case study in Section 8, we show the significant differences in energy and performance variability. The case for parallel applications is presented in Section 9. Other issues related to energy metering are discussed in Section 10. Section 11 draws the main conclusions of this work.

2. BACKGROUND AND RELATED WORK

2.1. Previous Energy Metering Approaches

With the increasing number of computing cores in processor architectures, the number and heterogeneity of the tasks that will coexist in a chip will increase. In this evolving scenario, it is of prominent importance to perform accurate per-task energy *metering* and *accounting* [Liu et al. 2013]. Given a workload composed by n tasks $T_1, T_2, \ldots, T_n$ running in a processor with n cores, *per-task energy metering* consists of tracking the energy that a given task, T_i, consumes during a given period of time. *Per-task energy accounting* consists of deriving for a given task T_i the energy that T_i would have consumed if it had run in isolation with a *fair share* of the hardware resources. As shown in Liu et al. [2013], energy accounting builds upon energy metering, so per-task energy metering is the first challenge to address.

In recent years, there has been an increasing interest for energy metering in different environments from data centers [Kansal et al. 2010; Bertran et al. 2012; Jimenez et al. 2011] to smartphones [Pathak et al. 2011; Carroll and Heiser 2010; Nokia 2012; Chung et al. 2011]. In those proposals, however, the focus is on providing accurate energy metering for single-core architectures, or multicore ones in which a single (multithreaded) application is executed. Those scenarios are relatively easy to handle because when an application is scheduled on the CPU, it is accounting all energy consumption of the

system (using a simple meter, for instance). In the case of smartphones, the proposals do not go beyond per-component energy measurement or energy estimation based on the execution time of a given task. In this article, however, we propose much more accurate techniques especially in the context of CMP and SMT processors, where applications dynamically share resources in nonobvious ways.

Many proposals [Bellosa 2000; Bircher and John 2012; McCullough et al. 2011; Howard et al. 2010] use Performance Monitoring Counters (PMCs) or system events (e.g., OS system calls) to break down the system energy consumption across its components (e.g., memory, processor). That is, authors develop power models that use a set of PMCs and predefined weights derived through correlation. In many cases, the results of the power model are compared against approaches utilizing circuit-based mechanisms such as current sense resistors. Although previous approaches have been shown to be very accurate in metering per-component and overall system energy consumption, they do not provide accurate per-task energy measurements, which is the focus of this article. It is our position that as processor design moves toward multi- and many-core processors with SMT cores, in which an increasing number of different applications simultaneously run on the same chip, providing accurate per-task energy metering becomes of paramount importance. Thus, our idealized reference per-task energy model and PTEM, its efficient implementation, cover this gap.

Recently, Shen et al. [2013] proposed a request-level OS mechanism to meter power consumption to each server request based on PMCs [Bellosa 2000]. The authors consider both active and maintenance power and attribute it to the responsible server requests. However, per-task energy estimates obtained with this approach cannot be obtained 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."* Moreover, to the best of our knowledge, no reference model has been reported for per-task energy metering.

In this article, we make the first proposal of (i) an idealized reference per-task energy measuring model identifying those parameters that must be tracked (many of them not tracked in existing processors) and (ii) hardware support to accurately measure per-task energy consumption in multicore processors with SMT cores narrowing the amount of information needed to still obtain accuracy levels close to those of the idealized reference model, but at an affordable hardware cost. Our approach is complementary to the one by Shen et al. [2013], since PTEM provides the hardware layer that delivers accurate per-task energy measurements (which Shen et al. lack), whereas their approach covers software-related implications of using the information provided by PTEM. Thus, this article presents the first reference per-task energy measuring model in multicore SMT processors, as well as an efficient implementation, PTEM, which is evaluated against the reference model.

2.2. Energy-Proportional Systems

In Figure 1, we showed an example of the energy variation across several workloads even if they are allocated the same amount of resources. These variations are already significant, and they will most probably increase in the future, as system manufacturers pay increasing attention to energy efficiency and energy-proportional computing [Barroso and Hölzle 2007].

A system is energy proportional if (i) it presents the maximum energy consumption when achieving the maximum performance, (ii) the energy consumption is close to zero when the system is idle, and (iii) the energy increases between these two extremes as performance increases as well. Although current systems are not yet fully, the trend is to move toward this kind of systems. In the presence of more energy-proportional

systems, static (and likely leakage) energy will decrease to some extent and dynamic energy will be the dominant source of energy consumption. Under this situation, energy consumption will depend more on the application activity and thus, considering per-task energy consumption, will be even more necessary.

2.3. Breaking Down Energy Consumption

Previous studies [Bircher and John 2012] have shown that the energy consumption of the processor can easily be around 50% to 60% of the total computing system energy consumption. In this article, we focus on that component and leave per-task energy-metering for the other resources as part of our future work. We break energy into its main three components: dynamic, static, and leakage.

Dynamic energy corresponds to the energy spent to perform those *useful* activities that circuits are intended to do, such as the energy spent in a register file to retrieve the contents of a particular register.

Static energy corresponds to the energy consumed due to *useless* activity not triggered by the program(s) being run. For instance, significant clocking power is consumed in idle blocks. Similarly, many SRAM arrays, such as cache memories, precharge some bitlines every cycle in order to speed up accesses. However, such activity is useless if no access occurs [Brooks et al. 2000]. Note that the energy consumed due to an access corresponding to a useless instruction (e.g., a misspeculated instruction) is considered as dynamic energy despite such activity is useless because the action has been triggered by the program(s) under execution. Breaking down dynamic and static energy is useful in our context, because it avoids mixing the energy consumed due to the activity triggered by the programs running and the energy that cannot be attributed to any program in particular if several of them are running.

Leakage corresponds to the energy wasted due to imperfections of the technology used to implement the circuit. Thus, leakage energy includes all energy wasted due to undesired current leaks and short circuits from supply to ground when transitioning gates from one state to another. Note that if circuits were implemented with *perfect* technology, no leakage power would be dissipated. This energy is referred to as static or leakage energy indistinctly in other works [Weste and Eshraghian 1988]. However, for the sake of clarity, we only use the term *leakage energy* to refer to this particular energy wasted due to imperfections of the technology.

3. IDEALIZED PER-TASK ENERGY METERING

This section presents an idealized utilization-based model for per-task energy metering. The result of this model is later used as a reference point for our models to measure per-task energy at an affordable hardware cost. For the sake of clarity, we assume a single voltage level and that energy consumption does not change with temperature. In Section 10, we show how to extend our models to consider the impact in energy consumption of multiple voltage levels and temperature ranges.

We assume a clustered multicore architecture where each cluster consists of a set of cores, each having core private data and instruction first-level caches, plus a shared on-chip second-level cache accessed through a shared bus (see Figure 2). We refer to such a cache as LLC. All clusters are connected to memory through a shared bus. We focus on the shared L2 caches, the core slice,[2] and the shared buses. The rest of the on-chip resources (e.g., I/O interface) have low contribution to total energy consumption [Nawathe et al. 2008], so we simply assume an even distribution of their energy consumption over running tasks, which has negligible impact on our estimation. If other

[2]In this article, core slice refers to the core plus the private L1 data and instruction caches.

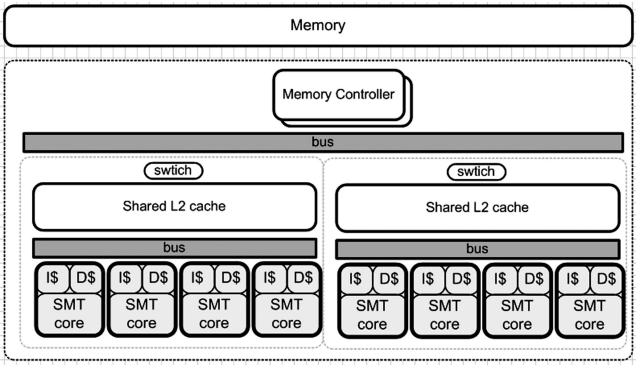

Fig. 2. Example of clustered architecture used in this article.

components have significant contribution to the total energy of the chip, energy metering should be extended accordingly following the same principles as for the components analyzed in this work.

3.1. Shared LLC

The dynamic energy consumption in the shared LLC for a given task i is proportional to the number of accesses. It can be computed as follows:

$$E^{LLC}_{dyn,total}(tk_i) = \sum_{k=1}^{K} \#action^{LLC}_k(tk_i) \times E^{LLC}_{action^{LLC}_k}, \quad (1)$$

where $E^{LLC}_{action^{LLC}_k}$ stands for the energy per LLC access of type k, which is assumed to be available in this idealized model; and $\#action^{LLC}_k(tk_i)$ stands for the number of LLC accesses of type k performed by the task i. Three main factors determine the access types we consider: whether an access reads or writes, hits or misses in LLC, and in the latter case whether it evicts a dirty line. The possible combinations are read hit, write hit, read miss replacing a dirty line, read miss replacing a nondirty line, write miss replacing a dirty line, and write miss replacing a nondirty line. Under each combination of these factors, the energy consumption of an access can change. Extending the model to consider other access types (e.g., invalidations) is trivial since we only need to multiply the energy consumed by each access type by the number of those accesses.

Static energy is consumed when resources are idle. We use cache occupancy as a proxy to measure static energy: we assume that those cache regions (lines) not occupied by a given task could be turned off so that they would not incur any energy consumption [Abella et al. 2005]. The total static LLC energy consumption for a task is obtained as follows:

$$E^{LLC}_{st,\ total}(tk_i) = Occ^{LLC}(tk_i) \times E^{LLC}_{st} \times IdleTime(LLC), \quad (2)$$

where $Occ^{LLC}(tk_i)$ stands for the average fraction of cache lines owned by task i, E^{LLC}_{st} corresponds to the static energy per cycle consumed by the LLC when no access is performed, and $IdleTime(LLC)$ stands for the number of idle cycles for the LLC (no access to LLC). E^{LLC}_{st} is assumed to be provided under the ideal model.

Leakage energy is proportional to the cache occupancy and can be easily computed as follows:

$$E^{LLC}_{leak,total}(tk_i) = Occ^{LLC}(tk_i) \times E^{LLC}_{leak} \times ExecTime(tk_i), \quad (3)$$

where E_{leak}^{LLC} stands for the leakage energy per cycle consumed by the LLC. This value is also an input parameter for the idealized model.

3.2. Core Slice

Ideal per-task core energy metering requires tracking per-task activity in all core hardware blocks (e.g., Reorder Buffer, Issue Queues) to count the number of accesses for each type. This would provide detailed information to accurately compute dynamic energy by multiplying the per-type access counts by the dynamic energy for each particular type of access (action):

$$E_{dyn,\ total}^{core}(tk_i) = \sum_{j=1}^{J} \sum_{k=1}^{K} E_{action_k}^{block_j} \times \#action_k^{block_j}(tk_i), \tag{4}$$

where $E_{action_k}^{block_j}$ is the energy per action of type k (e.g., read) in block j (e.g., register file), which is assumed to be known; and $\#action_k^{block_j}(tk_i)$ stands for the number of such actions on such block performed by task i. This applies to both single- and multithreaded (e.g., SMT) cores. In addition, J and K stand for the total number of blocks in the core and types of actions (e.g., read, write, flush), respectively.

Static energy is measured in all of those blocks having nonnegligible energy consumption when no action is performed. Blocks can be classified into two categories depending on whether they allocate entries to tasks. Occupancy Blocks or *oblocks* allocate entries to tasks, and hence their static energy can be split based on the occupancy (e.g., precharge energy of first-level caches). Conversely, in resources without memory or *eblocks,* no entries are allocated, and hence static energy can be evenly distributed (e.g., issue queue selection when there are no ready instructions). Static energy is then computed as follows:

$$E_{st,\ total}^{core}(tk_i) = \sum_{x=1}^{ExecTime(tk_i)} \left(\sum_{j=1}^{J} \frac{E_{st}^{eblock}(x)}{\#Tk(C_k)} + \sum_{l=1}^{L} Occ^{oblock}(tk_i) \times E_{st}^{block_l}(x) \right), \tag{5}$$

where L stands for the number of *oblocks*, J for the *eblocks*, and $Occ^{block_l}(tk_i)$ for the average occupancy of block l by each task; $E_{st}^{block_l}(x)$ stands for the static energy consumed by idle ports or in idle cycles of block l in cycle x; and $\#Tk(C_k)$ stands for the number of tasks in core C_k.

Leakage energy can be easily tracked because it will be roughly constant throughout all execution. If the core is single threaded, then it is trivial to identify the owner of such energy. However, if the core is multithreaded, the occupancy per task in each of the blocks must be tracked to properly distribute leakage energy, as shown in the following equation:

$$E_{leak,\ total}^{core}(tk_i) = \sum_{j=1}^{J} Occ^{block_j}(tk_i) \times E_{leak}^{block_j} \times ExecTime(tk_i), \tag{6}$$

where $Occ^{block_j}(tk_i)$ stands for the average occupancy of block j by task i; and $E_{leak}^{block_j}$ stands for the leakage per cycle of block j, which is assumed to be available.

3.3. Shared Bus

Ideal per-task bus energy metering requires tracking per-task accesses. Analogously to the case of the LLC, there are different types of accesses with different dynamic energy consumption. For instance, if a cache line is sent over the bus, the energy consumed

is higher than if just an address is sent, either because the cache line communication sends more bits simultaneously or because it requires several consecutive transactions to send all data over a bus narrower than a cache line. This would provide detailed information to accurately compute dynamic energy by multiplying the per-type access counts by the dynamic energy for each particular type of access (action):

$$E^{bus}_{dyn,\ total}(tk_i) = \sum_{k=1}^{K} E_{action_k} \times \#action_k(tk_i), \tag{7}$$

where E_{action_k} is the energy per action of type k (e.g., cache line communication), which is assumed to be known; $\#action_k(tk_i)$ stands for the number of such actions performed by task i; and K stands for the types of actions. Note that different actions and energy per action values may be used for different buses, such as the intracluster bus connecting cores to their LLC and the intercluster bus connecting cores to memory. Nevertheless, the same principle applies to compute dynamic energy.

Leakage energy cannot be attributed to any particular task in the cores (tasks do not have any type of bus occupancy), so we evenly distribute it across all those tasks that could use the particular bus: tasks in the cluster for intracluster buses and tasks in the whole chip for the intercluster bus:

$$E^{bus}_{leak,\ total}(tk_i) = \frac{E^{bus}_{leak} \times ExecTime(tk_i)}{\#Tk(BUS_k)}, \tag{8}$$

where E^{bus}_{leak} is the leakage energy per cycle of the bus, which is assumed to be known; and $\#Tk(BUS_k)$ stands for the number of tasks in using bus BUS_k. Note that bus energy is dominated by dynamic and leakage energy [Kumar et al. 2005] due to wiring, repeaters, and latches, whereas static energy is negligible. We evenly distribute static energy over tasks.

4. PTEM HARDWARE SUPPORT

Implementing the exact computation of the *idealized* energy model is expensive—if at all feasible—due to the large number of events to be tracked and the frequency at which they must be tracked. Therefore, we propose PTEM, our per-task energy metering approach that trades off energy metering accuracy and implementation complexity.

4.1. Hardware Support for the LLC

The ideal model for the LLC tracks two main per-task parameters: access (activity) counts per access type and cache occupancy. Our simplified PTEM model for the LLC relies on the fact that LLC accesses are not frequent, so they can be tracked with full accuracy. Conversely, tracking cache occupancy, which is required for static and leakage energy estimation, would require counting how many cache lines each task owns every cycle, which is expensive. Tracking the ownership of cache lines requires (i) tagging each cache line with a *task id*; (ii) keeping a counter per task with the number of owned cache lines (*instant counter*); and (iii) updating such counters on a replacement based on the ownership of the evicted and fetched cache lines, increasing the counter of the owner of the fetched line and decreasing the one of the owner of the evicted cache line.

In general, LLC access patterns and occupancy do not change abruptly. Similarly, the occupancy per set is quite homogeneous for any particular program [Moreto et al. 2008]. Therefore, we propose sampling the LLC occupancy in two different "directions." First, only some cache sets will be monitored, so they will be the only ones for whom cache line ownership will be tracked. In order to avoid clustering effects due to contiguous allocation of data in memory for any particular task, sampled sets are

located at a particular stride (e.g., only those sets whose x lowermost index bits are zero are monitored). How many x lowermost bits are considered depends on the desired sampling granularity. Second, the counters accumulating instant occupancy are not updated every cycle, but at a lower frequency.

For instance, for an LLC with 1,024 cache sets, eight ways per set and a processor with 8 cores, cache sets can be sampled at a granularity of 1 out of 16, and time sampling occurs once every 256 cycles. In this case, the overhead of the LLC mechanism would be as follows:

—8 instant counters (Occ_{inst}^{LLC}) of 10 bits each for tracking instant occupancy (1,024 sets $\times$ 8 ways/16 sample granularity = 512 lines sampled, so 10 bits are needed).
—512 3-bit owner identifiers for the 512 tracked cache lines. Note that all cache lines in the sampled sets always have an owner for energy metering purposes. Thus, on a context switch, the task being scheduled in becomes the owner of the cache lines used by the task being scheduled out (using the same hardware context, or CPU index).
—8 cumulated occupancy counters (Occ_{cum}^{LLC}) of 48 bits able to track the occupancy during $2^{48} \times 2^8 = 2^{56}$ cycles (48-bit counters and 256 cycles sampling frequency).

We assume that the number of cycles that a program takes to run is measured by an existing PMC of the processor. Based on this hardware support, LLC occupancy is obtained as follows:

$$Occ^{LLC}(tk_i) = \frac{Occ_{cum}^{LLC}(tk_i) \times SmpFreq \times SmpSets}{\#Sets^{LLC} \times ExecTime(tk_i)}, \tag{9}$$

where $SmpFreq$ is the sampling frequency (256 cycles in the example), $SmpSets$ is the set sample granularity (16 in the example), and $\#Sets^{LLC}$ is the number of total cache sets (1,024 in the example). The impact of sampling in both time and sets is later analyzed in the evaluation section.

4.2. Hardware Support for the Core

Current processors (e.g., the IBM POWER7 [Floyd et al. 2011; Huang et al. 2012]) can estimate the energy consumed by each core (even for SMT cores) based on a model that uses different PMCs, voltage, frequency, and temperature as proxy. However, solutions to accurately distribute core energy across tasks in SMT cores have not been developed, although, in fact, multicores with SMT cores are becoming quite common [Floyd et al. 2011; Singhal 2008].

A real per-task core energy metering cannot be done with the ideal model presented beforebecause this models tracks too many events and the occupancy of many blocks. Instead of such a bottom-up model, PTEM builds a top-down model. Under this top-down model, during the execution of a workload, we first break down the energy consumed into its main components, namely, dynamic, static, and leakage, and in a second step, we break down the energy of each component per task.

Step 1: Deriving static, dynamic, and leakage energy components. We start determining the maximum (E_{max}^{core}) and minimum energy (E_{min}^{core}) consumption in a given time interval. The core maximum energy consumption, E_{max}^{core}, can be determined by running a high-power benchmark, a.k.a. power virus [Naffziger et al. 2005]. E_{max}^{core} can be decomposed as follows:

$$E_{max}^{core} = MaxDynE^{core} + LeaE^{core}, \tag{10}$$

where $MaxDynE^{core}$ is the maximum dynamic energy of the core, and $LeaE^{core}$ is the leakage energy of the core that can be obtained by measuring core power when the core is in *halt mode*. In this formula, we assume that all blocks are fully used and

thus no static energy is consumed. In reality, there will be still some static energy, but its relative weight with respect to dynamic energy is negligible in a maximum energy scenario, so the loss of accuracy introduced by such an assumption is rather low.

The core minimum energy consumption, E_{min}^{core}, can be obtained running a low-power benchmark comprised, for instance of *no-ops*. E_{min}^{core} can be decomposed as follows, where $MaxStaE^{core}$ is the maximum static energy of the core:

$$E_{min}^{core} = MaxStaE^{core} + LeaE^{core} \tag{11}$$

In this formula, we assume that all blocks are idle so that no dynamic energy is consumed. Under that scenario, all activity in the core is static energy, as this activity is not produced by tasks activity. This is the scenario in which the static energy is the highest, $MaxStaE^{core}$. From Equations (10) and (11), we can derive $MaxDynE^{core}$ and $MaxStaE^{core}$.

Let's assume that the energy consumed by a workload during an interval j is $E_j^{core} = LeaE^{core} + DynE_j^{core} + StaE_j^{core}$. In order to determine which fraction of E_j^{core} is static, dynamic, and leakage, we proceed as follows. Leakage is roughly constant in all runs, so we take the value derived earlier, $LeaE^{core}$.

We assume that all idle blocks have the same static energy consumption when idle with respect to their dynamic energy consumption. That is, for all blocks, the relation between static and dynamic energy is obtained as $StaDynRatio = \frac{MaxStaE^{core}}{MaxDynE^{core}}$. Hence, the static energy for each block is $StaDynRatio$ of its dynamic energy.

During the execution of a workload in a given interval, a fraction of the resources will perform useful activity, thus consuming dynamic energy in the interval ($DynE_j^{core}$). The remaining resources do not perform any useful activity consuming static energy. The difference $MaxDynE^{core} - DynE_j^{core}$ provides the amount of dynamic energy not consumed in the execution of the workload with respect to the scenario in which the dynamic energy is maximum. The static energy $StaE_j^{core}$ is a fraction of that difference: $StaE_j^{core} = (MaxDynE^{core} - DynE_j^{core}) \times StaDynRatio$.

Overall, E_j^{core} can be derived as follows:

$$\begin{aligned} E_j^{core} &= LeaE^{core} + DynE_j^{core} + StaE_j^{core} \\ &= LeaE^{core} + DynE_j^{core} + \left(MaxDynE^{core} - DynE_j^{core}\right) \times StaDynRatio, \end{aligned} \tag{12}$$

where only $DynE_j^{core}$ is unknown and can, therefore, be derived.

Step 2: Breaking down static, dynamic, and leakage energy components per task. Per-task energy distribution is done as follows:

—*Dynamic energy*. Since tracking all events in the core is unaffordable, we use a simplified model based on the number of instructions fetched per task.
—*Static energy*. Most static energy in the core comes from register files and issue queues due to their large number of ports and high static energy consumption per port. Such energy cannot be attributed to any particular task, so we evenly split static energy across tasks.
—*Leakage energy*. Leakage energy mainly comes from first-level (L1) caches, and their occupancy correlates quite well with the occupancy of some other blocks (e.g., branch predictor tables, translation lookaside buffers). Thus, we track task occupancy in L1 caches. We need the same hardware support as in the LLC. We consider that L1 data and instruction cache occupancies have the same weight.

Table I. PTEM Hardware Requirements

Block	Energy Figures	Extra Logic
LLC	E^{LLC}_{action}, E^{LLC}_{st}, E^{LLC}_{leak},	$\#action^{LLC}_k(tk_i)$, $Occ^{LLC}_{inst}(tk_i)$, $Occ^{LLC}_{cum}(tk_i)$, $IdleTime(LLC)$, LLC Cache line owner's table
Core	E^{core}_{max}, E^{core}_{min}, $LeakE^{core}$	$InstFetch$, $InstFetch(tk_i)$, $Occ^{IC}_{inst}(tk_i)$, $Occ^{IC}_{cum}(tk_i)$, $Occ^{DC}_{inst}(tk_i)$, $Occ^{DC}_{cum}(tk_i)$, IC Cache line owner's table, DC Cache line owner's table, $E^{core}_{total}(tk_i)$
Intracluster bus	E^{inbus}_{action}, E^{inbus}_{leak}	$\#action^{inbus}_k(tk_i)$
Intercluster bus	E^{outbus}_{action}, E^{outbus}_{leak}	$\#action^{outbus}_k(tk_i)$

Therefore, task energy in the core is measured as follows for interval j:

$$DynE^{core}_j(tk_i) = DynE^{core}_j \times InstFetch_j(tk_i)/InstFetch_j \tag{13}$$

$$StaE^{core}_j(tk_i) = StaE^{core}_j/\#Tk \tag{14}$$

$$LeaE^{core}_j(tk_i) = LeaE^{core}_j \times \frac{Occ^{IC}(tk_i) + Occ^{DC}(tk_i)}{2}, \tag{15}$$

where $InstFetch_j$ and $InstFetch_j(tk_i)$ are the total and task i fetched instructions in interval j, respectively; and $Occ^{IC}(tk_i)$ and $Occ^{DC}(tk_i)$ stand for the task i occupancy in the data and instruction caches. Then, we only need to cumulate the energy of the task across all intervals:

$$E^{core}_{total}(tk_i) = \sum_{j=0}^{\frac{ExecTime(tk_i)}{SmpFreq}} \left(DynE^{core}_j(tk_i) + StaE^{core}_j(tk_i) + LeaE^{core}_j(tk_i)\right) \tag{16}$$

4.3. Hardware Support for the Buses

The ideal model for the buses only needs to track access (activity) counts per access type per task. Our simplified PTEM model for the buses relies on the fact that analogously, as for the LLC, bus accesses are not frequent, so they can be tracked with full accuracy. In addition, leakage energy is tracked trivially by considering how many cycles each thread runs and how many threads share each bus.

4.4. Putting It All Together

PTEM has some hardware overhead, as it requires setting up a reduced set of additional counters similar to those PMCs currently available in most high-performance processors. PTEM support, analogously to PMCs, does not interfere with the execution of programs since it is not in any critical path.

Table I summarizes the energy figures required from the chip vendor and the extra logic (counters, tables) that must be set up. Energy figures can be either obtained by running appropriate benchmarks or estimated using test chips or power models. Note that the (tk_i) suffix in some counters indicates that they must be replicated for each task. Analogously, *action*, in the case of the LLC, stands for the six different LLC actions considered in this article: read/write hit, read/write miss (no dirty line replaced), read/write miss (dirty line replaced), and for the two different bus actions considered

in this article in the case of the buses: address communication and cache line communication. *Inbus* and *outbus* refer to the intra- and intercluster buses, respectively, in the table.

Regarding the software interface, the OS is responsible for keeping track of the energy consumed by every task in the system. PTEM exports a special register, called Energy Metering Register (EMR), that acts as an interface between PTEM and the OS. The OS can reset the EMR (typically when a task is scheduled in) and access that register for collecting the energy estimates made by PTEM (typically when a task is scheduled out). In general, these actions happen during context switches. In the event of a context switch, the OS reads the EMR using the hardware-thread index (or CPU index) of the scheduled-out task (T_{out}). Then, the OS aggregates the energy consumption value received in the *task struct* of T_{out}.

Right after the new task (T_{in}) is scheduled in, the LLC and L1 caches still hold some lines belonging to T_{out}. These lines are tagged with the same identifier as the one assigned to T_{in}. Although PTEM will attribute static and leakage energy consumption to T_{in}, we have empirically observed that this occurs during less than 1 million cycles (often much less), since cache lines belonging to T_{out} are quickly replaced and thus evicted from LLC. If the processor operates at 2GHz, 1 million cycles elapse in 0.5ms.Context switches, instead, occur at much higher granularity. The OS quanta can vary from aggressive values (e.g., 4ms) to more conservative ones (e.g., 100ms) for common Linux and Windows implementations. Moreover, on a context switch, many tasks being run may not be scheduled out since the number of hardware contexts increases in pace with the number of cores. Thus, only tasks with lowest priority are expected to be scheduled out, which further reduces effective context switch frequency for most of the tasks.

The time that the OS spends working on behalf of a given task, for instance serving a system call, is attributed to the caller task. Similarly, the energy consumed by the OS on *housekeeping* activities is evenly attributed to all running tasks. Nevertheless, if other policies attributing OS energy to tasks are devised, then PTEM provides the hardware support needed for that purpose.

5. 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]. Wattch-like power models are built on top of the CACTI 6.5 simulation tool [Muralimanohar and Balasubramonian 2009]. CACTI is a flexible tool modeling delay, energy (dynamic and leakage), and area of cache memories and SRAM-based arrays. Power models for functional units have been updated to use modern designs.

We assume a clustered multicore architecture where each cluster consists of a set of cores, with each core having private data and instruction first-level caches, plus a shared on-chip second-level cache accessed through a shared bus. We refer to such cache as LLC. Details about the configuration can be found in Table II. All clusters are connected to memory through a shared bus (see Figure 2). Several studies show that hierarchical bus configurations scale quite easily to large systems and provide a good area-performance trade-off, while retaining many of the advantageous features of simpler bus arrangements [Salminen et al. 2007]. In the same line, other studies show that bus-based networks can significantly lower energy consumption and simplify network protocol design and verification with no loss in performance [Udipi et al. 2010].

We consider a processor configuration with clusters consisting of 4 cores, with those cores being two-threaded SMT (4C2S). We first perform an intracluster evaluation for this 4C2S setup to facilitate understanding of PTEM behavior. Then, for the full

Table II. Processor Configuration

Parameter	Description
Chip details	
Cluster count	1, 2, 4, and 8
Core count	4 cores per cluster; 1-, 2-thread SMT
Supply voltage	1.0V
Technology	65nm
Core details	
Core type	Out-of-order
Fetch, decode, issue, commit bandwidth	2 instr/cycle
Branch predictor	Hybrid 256B Gshare
Branch target buffer	1,024 entries, 4-way
Return address stack	256 entries
Reorder buffer size	128 entries
Issue queues size	32/32/32 entries for INT/FP/Load-store queues
Register file	80 INT, 80 FP
Functional units	2 INT ALU (1 cyc), 1 mult (4 cyc), 1 div (7 cyc) 1 FP ALU (6 cyc), 1 mult (6 cyc), 1 div (17 cyc)
Instruction L1	32KB, 4-way, 32B/line (2 cycles hit)
Data L1	32KB, 4-way, 32B/line (2 cycles hit)
Instruction TLB	256 entries fully associative (1 cycle hit)
Data TLB	256 entries fully associative (1 cycle hit)
Shared L2 Cache	
Unified L2	2MB, 16-way, 64B/line (3 cycles hit, 300 cycles miss)

processor evaluation, we consider setups of four and eight clusters, with two-way SMT 4-core (4C2S) clusters. Thus, 32- and 64-threaded setups are considered, respectively.

Benchmarks. In addition to traces from a real HPC application, detailed in Section 9, we use traces collected from the whole SPEC CPU2006 benchmark suite using the reference input set. Each trace contains 100 million instructions, selected using the SimPoint methodology [Sherwood et al. 2001]. Using these benchmarks, we generate different workloads with different numbers of benchmarks.

Running all N-task combinations is increasingly time consuming, as the number of combinations is too high. 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 three workload types denoted *I*, *M*, and *X*, depending on whether all benchmarks in a cluster belong to group *ILP*, *MEM*, or a combination of both.

We generate eight workloads per group and processor setup. Benchmarks in each workload are randomly picked out from all 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, because the random selection of benchmarks is always performed out of the corresponding (original) group of benchmarks.

Metrics. In order to evaluate the accuracy of PTEM, as a reference, we always use the estimates that we would obtain with the idealized model. In each experiment,

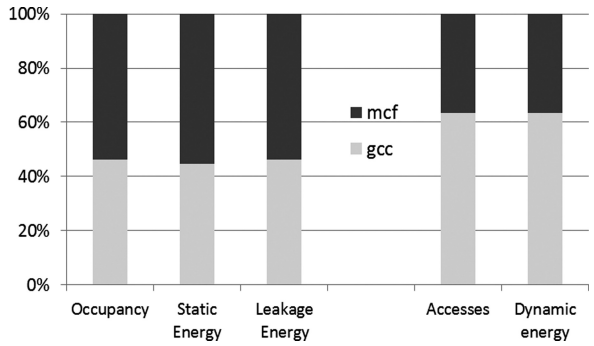

Fig. 3. Per-task LLC cache energy breakdown and access/occupancy rates when executing *mcf* and *gcc* in a single-threaded 2-core configuration.

we measure the *off estimation* or *prediction error* of each model with respect to the idealized model, which is computed as follows:

$$PredictionError = \left| 1 - \frac{EnergyEstimate_{model}}{EnergyEstimate_{ideal}} \right| \tag{17}$$

6. INTRACLUSTER EVALUATION

We evaluate the accuracy of our hardware support for per-task energy metering incrementally by analyzing the accuracy at the intracluster level. Once we analyze the accuracy of the PTEM models for the cache and SMT core, the next section shows the results when we scale the number of cluster to sum up a total of 16/32 cores (32/64 threads). Due to the relatively low energy contribution of buses, intracluster bus energy is reported as part of the LLC energy.

6.1. Utilization-Based Per-Task Energy Consumption

The key idea of our per-task energy metering approach is to make the energy attributed to a task proportional to each resource—in particular, to its *activity* and the *occupancy* of a given resource. If both activity and occupancy are accurately measured, the energy consumption can be accurately attributed to each running task.

Figure 3 shows the fraction of LLC energy consumption attributed to each benchmark in a 2-core workload (*gcc*+*mcf*) by using our ideal model presented in Section 3.

We observe that the activity does not necessarily reflect the occupancy of the LLC. In the figure, we can see that *gcc*, with 63.5% accesses, occupies less than 46.2% of LLC lines. That shows that a given workload may have very different consumption profiles in terms of dynamic energy versus static and leakage energy. Therefore, it is important to measure both activity and occupancy in order to improve the estimation accuracy. For instance, let us look again at the *gcc* case. If we estimate the energy only proportionally to the activity, LLC energy will be significantly overestimated for *gcc* and underestimated for *mcf*.

6.2. PTEM Energy Estimation

In this section, we show the accuracy of the models presented in Section 3 for the core and the LLC at cluster level. In particular, we measure the off estimation of each model with respect to the idealized model. We include the Evenly Split (ES) model, which uniformly splits the energy among all running tasks regardless of their occupancy and activity in the processor resources. This is indeed the common approach in current methods only considering execution time.

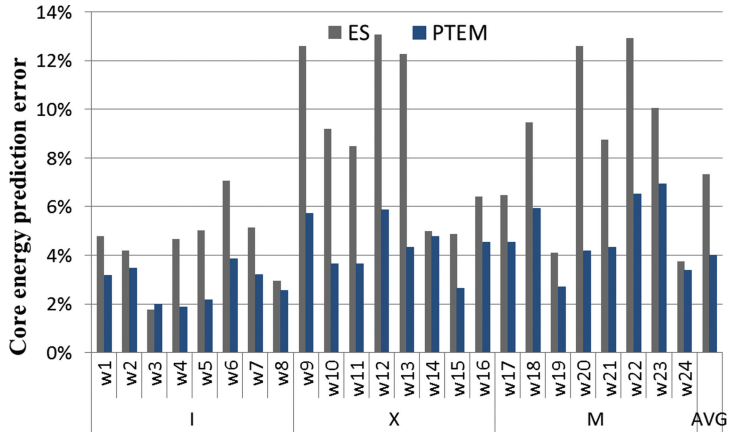

Fig. 4. Per-task core energy prediction error rate (C4S2).

Table III. Maximum Per-Task Prediction Error

Core			
	I	X	M
PTEM	8.8%	9.6%	10.2%
ES	11.9%	28.3%	21.9%
LLC			
	I	X	M
PTEM	25.6%	4.0%	4.4%
ES	1112.6%	3593.8%	62.0%
Cluster			
	I	X	M
PTEM	6.6%	9.2%	7.5%
ES	25.8%	58.5%	23.6%

Core Energy Consumption Prediction. Figure 4 shows the prediction accuracy for the core under the setup C4S2. Each bar shows the average error of all eight benchmarks in the workload.

In general, PTEM clearly outperforms ES by providing tighter energy predictions. In particular, PTEM incurs a prediction error of up to 6.9% across workloads, whereas it is higher than 13% for the ES model. Predictions are more accurate for *I* workloads due to the highly homogeneous behavior of programs. Irregular workloads in *X* and *M* groups (some benchmarks are more memory bound than others in the *M* group) lead to slightly higher error for PTEM and larger error for the ES model. This can also be seen when comparing the maximum error across individual tasks in the workloads (see Table III). PTEM maximum error is highly constant across workload types (9% to 10%), whereas ES model error is particularly high for *X* and *M* workloads (28% and 22%, respectively).

LLC Energy Consumption Prediction. Figure 5 shows the effect of sampling sets and period on the average LLC energy prediction accuracy for a 4-core configuration. The y-axis represents the sample period measured in processor cycles (e.g., 10K stands for 10,000 cycles). The x-axis is the sampling set configuration. For instance, *1e8* means that we sample one set every eight sets.

We observe that the curve has a higher slope in the x-axis (set sampling). For instance, for a sampling distance of 10K cycles, the prediction error rate raises from less than

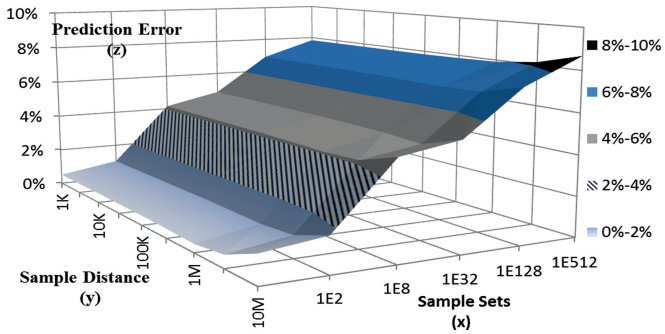

Fig. 5. Per-task LLC cache energy prediction with sample set and period in a 4-core configuration.

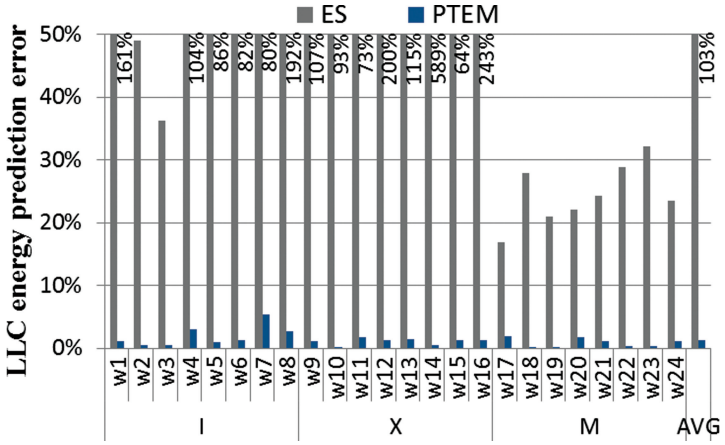

Fig. 6. Per-task LLC cache energy prediction error rate (C4S2).

1% to almost 8% as the sample set reduces from 1e1 to 1e512 sets. Instead, the sample period (y-axis) has limited effect on accuracy. With 1e8 sampled sets, the prediction error only raises 0.2% as the period increases from 1K to 10M cycles. Considering that the hardware cost of set sampling varies significantly, we choose a moderate cost configuration in which we use a 1e2 and 10K cycle sampling period. This is the configuration that we use to measure the energy per task in the LLC in the following sections.

Figure 6 shows the LLC prediction error of each model under the C4S2 setup. Prediction error corresponds to the average error across benchmarks in each workload. We observe that PTEM largely outperforms the ES model in terms of accuracy for all workloads and processor setups.

The ES model is highly inaccurate in general—more than 103% on average. The ES model accuracy is worse for *I* and *X* workloads due to the highly heterogeneous memory behavior of the tasks. In fact, even in *I* workloads, behavior is highly heterogeneous because the relative LLC access frequencies and occupancies are very different across tasks. ES accuracy improves for *M* workloads where LLC occupancy and access frequency are more homogeneous. Our PTEM model, in contrast, has a considerably low prediction error—less than 2% on average. Further, as shown in Table III, maximum error across all tasks for PTEM is 25.6% for *I* workloads because their low LLC utilization may make spatial sampling to experience some error. However, as long as *M* tasks

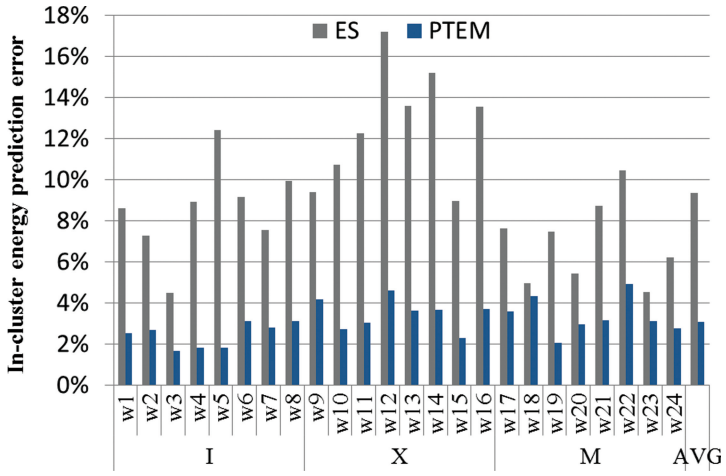

Fig. 7. Cluster per-task energy prediction error (C4S2).

are in place (X and M workloads), PTEM accuracy is very high (maximum error is always less than 4.5%). On the other hand, ES model error is huge (more than 3,000%), especially for I and X workloads due to the highly heterogeneous memory behavior of the tasks in the workloads.

Cluster Energy Consumption Prediction. Next, we show per-task energy metering accuracy at cluster level, including core and LLC energy. Figure 7 shows the average prediction error in each workload for a cluster consisting of four two-way SMT cores. First, we observe that prediction error for the whole cluster is very similar to that of the cores only (see Figure 4). This is so because the LLC energy contribution is typically in the range of 15% to 20% due to the high activity of the cores (8 threads running). Therefore, core prediction error dominates the overall prediction error. As expected, the ES model obtains worse results than PTEM in all workload groups, with an average of more than 10%. The prediction error for PTEM is less than 3% on average across all workloads. Furthermore, we observe that the ES model error grows for X workloads since different threads perform highly heterogeneous activities. ES model average error is higher than 17% for one of the workloads. Instead, PTEM error remains quite stable across workloads and never exceeds 4.5%.

Per-benchmark data in each workload show that the maximum off estimation that PTEM produces is 9.2% for one of the benchmarks in the X workloads (see Table III; recall that we use 8-benchmark workloads and evaluate 24 different workloads, counting 192 benchmarks in total). For homogeneous workloads (I and M), the maximum error observed is only 7.5%. Instead, the maximum error for the ES model is 58.5%. Maximum error is lower for homogeneous workloads but is still in the order of three to four times that of our PTEM model.

6.3. PTEM Energy and Area Overhead

PTEM requires few hardware counters to track LLC and core and bus activity, together with small arrays tracking the ownership of some cache sets in the LLC and L1 caches. For the sake of consistency, the energy of those components has been modeled using CACTI. In order to model counters, components such as internal cache buffers have been used, as they are comparable to latches in the pipeline.

Results for the 4-core two-way SMT configuration show that the total energy overhead for PTEM is less than 0.3%. Most of the overhead is due to the dynamic energy

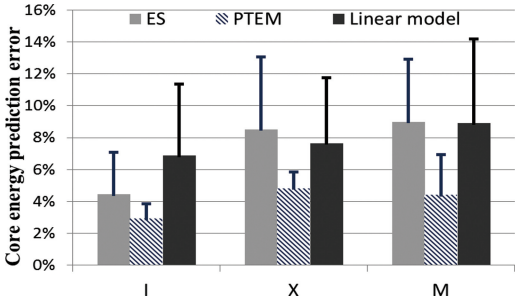

Fig. 8. Core per-task energy prediction (C4S2) with ES, PTEM, and the linear model, including the average error and maximum error.

of the ownership id arrays in LLC and L1 caches. Relative overheads do not change noticeably for different core counts. In fact, the relative overhead slightly decreases as the number of cores increases, which proves that PTEM scales well.

We have obtained the area overhead using CACTI with the following assumptions: LLC cache occupies 50% of the area in a 8-core configuration, counter bitcells have the same size as input/output buffers in caches (so they are large), and ALUs performing power computations use low-cost designs such as iterative multipliers and dividers (their latency is not critical, as they are seldom used). We consider SMT cores, as they require more bits to track ownership and more counters to track per-task activity. Overall, we obtained that total area overhead is 0.49% (4 cores), 0.63% (8 cores), 0.75% (16 cores), and 0.82% (32 cores), proving that PTEM area cost is rather low. The area breakdown for the 32-core configuration is 0.20% LLC, 0.48% DL1+IL1, 0.09% core without DL1/IL1, and 0.05% bus. Similarly, the breakdown for the 4-core configuration is 0.22% LLC, 0.20% DL1+IL1, 0.04% core without DL1/IL1, and 0.03% bus. Thus, those arrays tracking the cache line ownership and the counters tracking per-task activities in caches account for most of the area overhead, which anyway is rather low.

Overall, PTEM imposes neither limitations on the number of threads that can be run simultaneously in the processor (low and scalable hardware overheads) nor limitations on the number of tasks that the OS can keep active simultaneously (a single counter per task needs to be tracked by the OS).

6.4. Linear Regression Models

Since linear regression models have been widely used to estimate core and system-level energy [Bellosa 2000; Bircher and John 2012; Shen et al. 2013], we also include it in our discussion. Although these models typically rely on existing PMCs, so no extra hardware support is needed, their accuracy is limited and highly depends on whether training workloads are similar to those at deployment.

Coefficients of the linear regression model are obtained using our idealized model as the reference model, because no other reference model exists. We provide the linear regression with all per-task event counters in our simulator, including number and type of instructions fetched, executed, committed, data and instruction cache hits and misses, and so forth, despite that PMCs may not exist for many of those events.

We have used a 4-core 2-thread SMT setup. The training set consists of a workload with eight benchmarks randomly chosen from the SPEC CPU2006 for each of the threecategories described before: *I*, *X*, and *M*. The evaluation workload consists of eight workloads generated analogously for each category.

As shown in Figure 8, the linear regression model performs worse than PTEM. Linear regression is less accurate than ES for *I* workloads and slightly more accurate for *X*

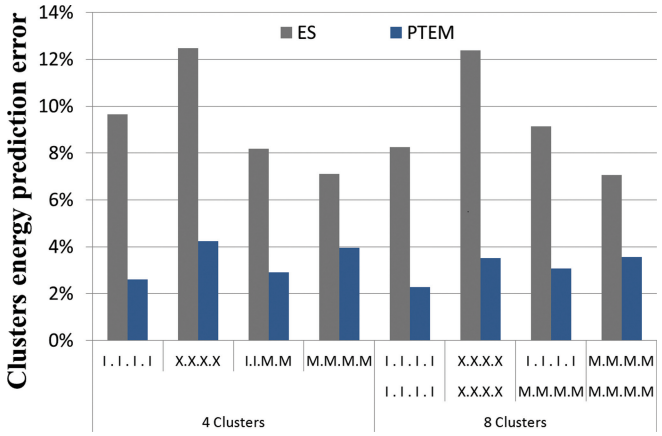

Fig. 9. System per-task energy prediction error.

and *M* workloads. The average error for the linear regression model is 7.8%, similar to that for ES. Furthermore, we have observed that maximum estimation error is higher for the linear regression model than for PTEM and ES. The reason for those large estimation errors for the linear regression model is twofold: (i) its dependence on the training set and (ii) the fact that PMCs do not take into account occupancy, which is the parameter determining per-task leakage and static energy in many components.

Finally, although not shown in the article, results for other components (e.g., LLC) show similar trends because of the same limitations pointed out for the core. For instance, Figure 3 shows the dependence of LLC leakage and static energy on occupancy rather than on accesses.

7. PTEM FOR HIGH CORE COUNTS

In this section, we evaluate the accuracy of our PTEM model for large multicores with four and eight clusters, counting 16 and 32 two-way SMT cores, respectively. For that purpose, we have run experiments with four different types of workloads: pure *I* workloads, pure *M* workloads, *X* workloads (with four *I* and four *M* tasks per cluster), and hybrid workloads where half of the clusters run pure *I* workloads and the other half run pure *M* workloads.

Memory bandwidth for the eight-cluster configuration has been increased by setting up two memory controllers instead of one able to issue memory commands in parallel as long as they do not conflict in any particular bank. This has been done in order to not overdesign memory bandwidth for the 4-core setup and to not underdesign memory bandwidth for the 8-core setup. The behavior of the different workloads is such that the relative execution time increase is low with respect to the single-cluster setup since little memory contention is suffered in *I* tasks, and the higher contention paid by *M* tasks is still low in relative numbers.

Results in Figure 9 show that PTEM achieves higher accuracy for pure *I* workloads and hybrid *I*-*M* workloads. This is so because, as shown previously, PTEM achieves higher accuracy for pure *I* clusters than for *X* or *M* clusters. Instead, pure *M* and *X* configurations show slightly higher prediction error. Nevertheless, the average error is low and, in the case with the largest core count (eight clusters), the average error is less than 3.5% regardless of the workload type. In the case of the ES model, prediction error is significantly higher than that of PTEM, being higher than 12% on average for *X* workloads. Other configurations show lower error since they mitigate the per-core

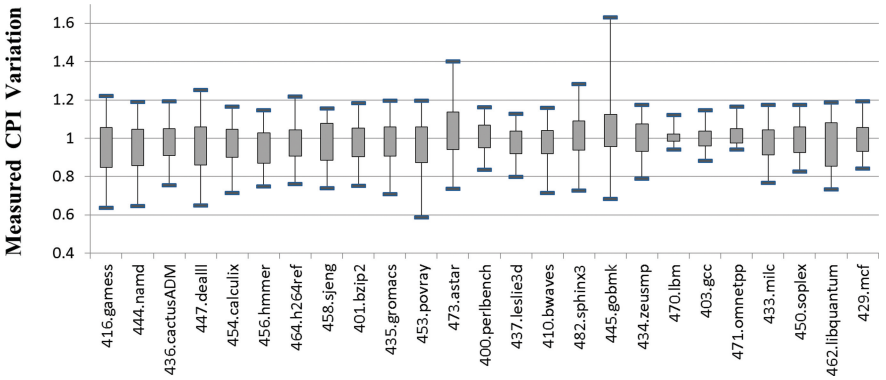

Fig. 10. Per-benchmark *CPI* variation across all two-task workloads in which the benchmark runs. Benchmarks sorted in increasing average *CPI*. Chart shows max, min, higher-quartile and lower-quartile values.

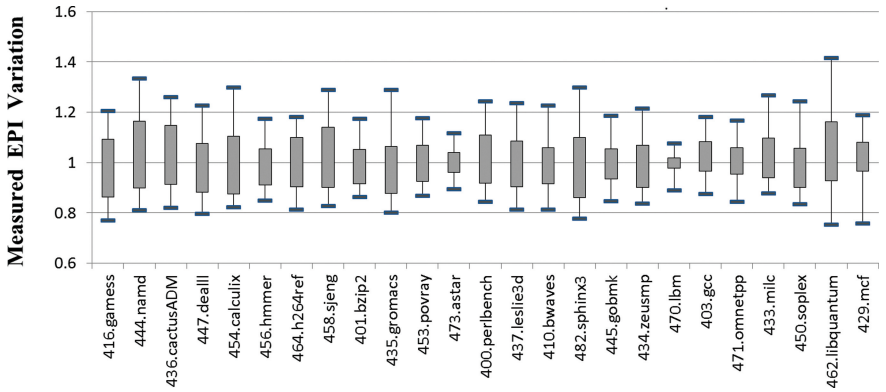

Fig. 11. Per-benchmark *EPI* variation across all two-task workloads in which the benchmark runs.

prediction error. Nevertheless, PTEM largely improves accuracy with respect to the ES model across cluster counts and workload types, and opposed to the ES model, PTEM error decreases as the cluster count increases.

8. CASE STUDY: CHARACTERIZATION OF ENERGY AND PERFORMANCE VARIATION

Interferences among co-running tasks when accessing shared hardware resources in a multicore (a.k.a. intertask interferences) result in different per-task performance depending on its co-runners [Fedorova et al. 2004]. In this section, we use our proposed PTEM model to show how the energy consumption of each task also significantly varies due to intertask interferences, and *prove that such energy consumption variation cannot be directly inferred from performance variation*.

We focus on two-task workloads, which we run in a two-way SMT core setup of our baseline configuration. We construct all possible pairs of benchmarks from SPEC CPU2006 suite, recording for each benchmark its energy consumption in each of the two-task workloads in which it runs. The variation that each benchmark suffers across each two-task workload is illustrated in terms of Cycles Per Instruction (CPI) in Figure 10 and in terms of Energy Per Instruction (EPI) in Figure 11. Results have been normalized with respect to the average *CPI* and *EPI*, respectively, for the sake of readability since *CPI* ranges between 1.03 and 11.36 cycles/instr, and

Table IV. Average *CPI* (cycles/instr) and *EPI* (nJ/instr) for All Benchmarks, Sorted from Lowest to Highest Average *CPI* from Left to Right and from Top to Bottom
MEM benchmarks are shown in bold font.

	416.gamess	444.namd	436.cactusADM	447.dealII	454.calculix
CPI	1.03	1.03	1.05	1.05	1.10
EPI	0.35	0.29	0.40	0.36	0.32
	456.hmmer	464.h264ref	458.sjeng	401.bzip2	435.gromacs
CPI	1.22	1.23	1.24	1.28	1.30
EPI	0.40	0.48	0.37	0.46	0.41
	453.povray	473.astar	400.perlbench	437.leslie3d	**410.bwaves**
CPI	1.34	1.34	1.41	1.42	1.46
EPI	0.41	0.50	0.44	0.50	0.55
	445.gobmk	482.sphinx3	434.zeusmp	**470.lbm**	**403.gcc**
CPI	1.51	1.51	1.60	2.19	2.58
EPI	0.49	0.50	0.56	0.92	0.89
	471.omnetpp	**433.milc**	**450.soplex**	**462.libquantum**	**429.mcf**
CPI	2.83	3.22	3.88	4.90	11.36
EPI	0.95	0.89	1.14	1.26	2.99

EPI between 0.29 and 2.99 nJ/instr. Benchmarks are sorted from lowest to highest *CPI*.

We observe that *CPI* variation mostly concentrates in the range [+20%, –40%] with respect to their average for most of the benchmarks, whereas *EPI* concentrates in the range [+30%, –20%]. Hence, in both cases, variations are significant; therefore, we can conclude that performance and energy consumption strongly depend on the co-runners. In terms of performance variation, *MEM* benchmarks (*mcf*, *milc*, *lbm*, *libquantum*, *soplex*, *gcc*, *bwaves,* and *omnetpp*) are among those with the lowest performance variation. For instance, *lbm* and *omnetpp*, both in the *MEM* category, are the ones exhibiting the lowest performance variation across all benchmarks.

However, in terms of *EPI*, this is not the case: typically, *EPI* variation for *ILP* benchmarks decreases, whereas *MEM* benchmarks have higher *EPI* variation than *CPI* variation. For instance, *libquantum*, which falls in the *MEM* category, is the benchmark exhibiting highest *EPI* variation. Analogously, *mcf*, *soplex*, *gcc,* and *omnetpp* also experience a significantly higher variation increase in terms of *EPI* than *CPI*. In contrast, *astar* has a significant variation in *CPI* but reduced variation in *EPI*. Thus, the relation between performance and energy variation is not obvious. We note that the two benchmarks in the middle of the x-axis, *astar* and *perlbench*, both of them being *ILP*, have opposite trends across metrics: *EPI* variation for *astar* is much lower than its *CPI* variation. Conversely, *EPI* variation for *perlbench* is much higher than its *CPI* variation.

We have also studied absolute *EPI* and *CPI* values, shown in Table IV. Values are sorted based on their *CPI*. We observe that *MEM* benchmarks have higher *CPI* than *ILP* ones, since they access memory more often and thus experience higher latencies. Few *ILP* benchmarks have higher *CPI* than some of the *MEM* ones. Such higher *CPI* for *MEM* benchmarks translates into higher average *EPI*. In fact, only *zeusmp* (*ILP*) has slightly higher *EPI* than one of the *MEM* benchmarks (*bwaves*). The main reason for the increased *EPI* of *MEM* programs is the fact that they execute longer and occupy more resource space, which translates into higher static and leakage energy. However, the particular interaction across different programs in shared resources leads to different behavior in terms of performance and energy, as it has been shown in Figures 10 and 11 in terms of *CPI* and *EPI* variability. Therefore, *performance cannot be used as a*

suitable metric to derive per-task energy consumption. Quite the opposite, these results confirm that our proposed energy metering technique, PTEM, is required to achieve accurate per-task energy metering.

9. CASE STUDY: LARGE-SCALE PARALLEL APPLICATIONS

9.1. Multithreaded Applications

In our per-task energy measuring approach, the energy accounted to each thread is saved into a special purpose register per thread, denoted EMR. Section 4.4 shows how the OS handles the EMR of each task.

The support required for PTEM in the case of multithreaded applications is simple. In fact, PTEM logic does not need to be changed. Changes are only needed on how the OS handles the EMR: the OS or the parallel runtime simply needs to aggregate the energy consumption estimates stored for all threads belonging to the same multithreaded application $Emeter_{App} = \sum_{i=1}^{N} EMR_i$, where N is the number of threads of the application. When a cache line is shared in the LLC across different threads, its energy (static and leakage) must be accounted once, either by splitting it across the threads sharing it or by attributing it to one of those threads. In particular, we identify as owner the thread fetching the cache line to the LLC. Whether this energy is attributed to one thread or another of the parallel application is irrelevant, since the energy of all threads will be finally aggregated to provide a single figure for the whole application. However, per-task energy can also be monitored individually and periodically during the execution so that such information can be later used to optimize the energy profile of the application. This is better illustrated in the next subsection through a particular example. The information provided helps in understanding the effects in terms of energy of unbalanced thread execution times.

9.2. Weather Research and Forecasting Model

In this section, we evaluate our energy metering mechanism with real traces from a parallel HPC application running on an actual supercomputer: `wrf`. The Weather Research and Forecasting (`wrf`) model [Michalakes et al. 2004] is a mesoscale numerical weather prediction system designed to serve both operational forecasting and atmospheric research needs. In this experiment, we use the nonhydrostatic mesoscale model dynamical core. Simulating all threads of the parallel MPI application implies a significant amount of simulation time, as these applications usually run for days or weeks on a supercomputer. We use an automatic mechanism to choose the most representative computation regions to be traced and simulated with a cycle-accurate simulator [Gonzalez et al. 2011]. This simulation methodology uses nonlinear filtering and spectral analysis techniques to determine the internal structure of the trace and detect periodicity of applications. Afterward, we use a clustering algorithm to determine the most representative computation bursts inside an iteration of the application.

We obtain four representatives for the five computation phases that compose the 64-thread MPI application. We have used these reduced trace files to feed the cycle-accurate architecture simulator described in Section 5. We simulate all threads sharing the LLC cache (four threads in this case study) in a CMP architecture (single-threaded cores). When a thread finishes executing, it waits until all other threads have also finished.

Figure 12 shows the evolution of the per-task energy breakdown in the CMP between two barrier communications. Note that energy components are stacked in the plot. At the beginning, thread 0 (Th0) consumes more energy than the other threads due to its higher activity (it behaves as an I task). Conversely, Th1, Th2, and Th3 behave as M tasks; therefore, their energy consumption is dominated by static and leakage energy.

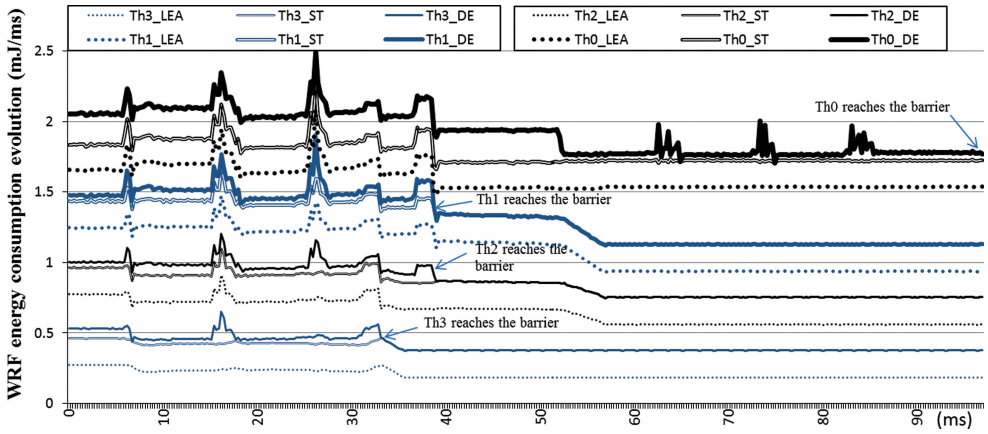

Fig. 12. Stacked power consumption evolution for *wrf* between two barriers.

Eventually, Th3 reaches the barrier and stops consuming dynamic energy. Th3 quickly loses its LLC lines, which decreases its leakage energy. Hence, Th3 energy consumption from this point onward corresponds to its core static and leakage energy. Behavior for Th1 and Th2 is analogous to that of Th3, but it takes longer for them to lose their LLC cache lines because they reach the barrier before 40ms and lose their LLC cache lines after 50ms. Th0, however, behaves as an *I* task for 52ms. Then it enters into an *M* phase, thus decreasing its dynamic energy. At this point, Th0 starts increasing its LLC occupancy, evicting Th1 and Th2 lines until it occupies the whole LLC after 57ms. This makes the leakage energy contribution of Th0 to grow noticeably.

Notice that our energy metering mechanism does not need to be aware of the synchronization among threads of a multithreaded task. For example, if a thread is busy waiting on a lock, even if it is not progressing during that time, the thread is using the processor and it will be metered accordingly. In contrast, if the thread goes to sleep until the lock is released, the core will go to low power mode and less energy will be metered to the thread.

10. VOLTAGE AND TEMPERATURE AWARE ENERGY METERING

Voltage and temperature influence energy consumption, so they cannot be neglected in general. IBM POWER7 [Floyd et al. 2011; Huang et al. 2012] power proxy is already aware of voltage and temperature, which are obtained through sensors. The power proxy scales dynamic, static, and leakage energy with constant factors associated to different voltage/temperature combinations. However, such proxy does not discriminate energy in a per-task basis, thus it cannot directly be used for PTEM.

Instead, a potential implementation for PTEM could track activities in a per-voltage and per-temperature basis, in such a way that the number of counters required matches the number of combinations of voltage and temperature ranges. For instance, if our chip can operate at 0.8V, 0.9V, and 1.0V, and temperature ranges considered are 320K–330K, 330K–340K, and 340K–350K degrees, then nine counters are required for each event to consider all combinations. Owner id tags in caches and occupancy counters will not need to be replicated (note that those arrays are responsible for most of the PTEM overheads).

Although voltage and temperature parameters may impact energy consumption of PTEM, their variability is expected to decrease with technology scaling and the increasing number of cores per chip. In particular, smaller geometries suffer from

process variations, which limit the minimum voltage that can be used [Bickford et al. 2008]. On the other hand, power efficiency and heat dissipation push for lower operating voltages. Thus, although dynamic voltage scaling techniques may still exist in the future, the range of voltages is expected to decrease, thus leading to fewer voltage levels. Temperature variation may be significant across the chip, but cores will become smaller with technology scaling, thus exhibiting lower in-core temperature variation due to the fact that meaningful temperature gradients occur at a nearly constant minimum distance [Donald and Martonosi 2006]. For instance, a difference of 1 degree canonly be observed at distances above 0.2mm, and cores may occupy less than $1mm^2$ in the near future. Similarly, LLC will remain in a narrow range of temperatures due to its relatively low activity. Moreover, maximum allowed temperature decreases due to technology scaling, because smaller devices age faster and aging has an exponential dependence on temperature.

11. CONCLUSIONS

In this article, we look at the challenges and opportunities associated with accurate per-task energy metering. As shown in this work, existing approaches based on an even distribution of energy across tasks are highly inaccurate. Therefore, we propose (i) a fair reference approach to distribute energy across tasks and (ii) an affordable implementation, PTEM, that tracks task activity and resource utilization at very low cost (below 0.3% energy overhead and 0.85% area overhead).

PTEM is shown to provide highly accurate per-task energy estimates with an average error of 3.1% for SMT multicore configurations and 2.1% for single-threaded multicore configurations. We further discuss the required changes, at both the hardware and the software levels, to provide such an accurate, yet implementable, per-task energy metering mechanism.

Finally, we show how to use PTEM in the context of parallel applications and show a case study where PTEM provides the required information to minimize the energy consumption of servers and cloud computing facilities.

REFERENCES

Abella, J., González, A., Vera, X., and O'Boyle, M. 2005. Iatac: a smart predictor to turn-off l2 cache lines. *ACM Trans. Archit. Code Optim. 2,* 1, 55–77.

Barroso, L. 2005. The price of performance. *ACM Queue 3,* 7, 48–53.

Barroso, L. and Hölzle, U. 2007. The case for energy-proportional computing. *IEEE Computer 40,* 12, 33–37.

Belady, C. and Malone, C. 2006. Data center power projections to 2014. *Proceedings of the Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronics Systems*, 439–444.

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

Bertran, R., Becerra, Y., Carrera, D., Beltran, V., Gonzílez, M., Martorell, X., Navarro, N., Torres, J., and Ayguadé, E. 2012. Energy accounting for shared virtualized environments under dvfs using pmc-based power models. *Future Gener. Comput. Syst. 28,* 2, 457–468.

Bickford, J., Rosner, R., Hedberg, E., Yoder, J., and Barnett, T. 2008. Sram redundancy - silicon area versus number of repairs trade-off. In *IEEE / SEMI Advanced Semiconductor Manufacturing Conference*. 387–392.

Bircher, W. and John, L. 2012. Complete system power estimation using processor performance events. *IEEE Transactions on Computers 61,* 4, 563–577.

Brooks, D., Tiwari, V., and Martonosi, M. 2000. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA*. 83–94.

Carroll, A. and Heiser, G. 2010. An analysis of power consumption in a smartphone. In *USENIX Annual Technical Conference*. 21–21.

CHUNG, Y.-F., LIN, C.-Y., AND KING, C.-T. 2011. Aneprof: Energy profiling for android java virtual machine and applications. In *ICPADS*. 372–379.

DONALD, J. AND MARTONOSI, M. 2006. Techniques for multicore thermal management: Classification and new exploration. In *ISCA*. 78–88.

FEDOROVA, A., SMALL, C., NUSSBAUM, D., AND SELTZER, M. 2004. Chip multithreading systems need a new operating system scheduler. In *11th ACM SIGOPS European Workshop*.

FLOYD, M., ALLEN-WARE, M., RAJAMANI, K., BROCK, B., LEFURGY, C., DRAKE, A., PESANTEZ, L., GLOEKLER, T., TIERNO, J., BOSE, P., AND BUYUKTOSUNOGLU, A. 2011. Introducing the adaptive energy management features of the power7 chip. *Micro, IEEE 31,* 2.

GONZALEZ, J., GIMENEZ, J., CASAS, M., MORETÓ, M., RAMÍREZ, A., LABARTA, J., AND VALERO, M. 2011. Simulating whole supercomputer applications. *Micro, IEEE 31,* 3, 32–45.

HAMILTON, J. 2009. Internet-scale service infrastructure efficiency. In *ISCA*. 232–232.

HOWARD, D., GORBATOV, E., HANEBUTTE, U., KHANNA, R., AND LE, C. 2010. Rapl: memory power estimation and capping. In *ISLPED*. 189–194.

HUANG, W., LEFURGY, C., KUK, W., BUYUKTOSUNOGLU, A., FLOYD, M., RAJAMANI, K., ALLEN-WARE, M., AND BROCK, B. 2012. Accurate fine-grained processor power proxies. In *45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'12)*. 224–234.

JIMENEZ, V., GIOIOSA, R., CAZORLA, F., VALERO, M., KURSUN, E., ISCI, C., BUYUKTOSUNOGLU, A., AND BOSE, P. 2011. Energy-aware accounting and billing in large-scale computing facilities. *Micro, IEEE 31,* 3, 60–71.

KALLA, R., SINHAROY, B., STARKE, W., AND FLOYD, M. 2010. Power7: Ibm's next-generation server processor. *Micro, IEEE 30,* 2, 7–15.

KANSAL, A., ZHAO, F., LIU, J., KOTHARI, N., AND BHATTACHARYA, A. 2010. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC)*. 39–50.

KOOMEY, J. 2011. Growth in data center electricity use 2005 to 2010. *Analytics Press*.

KUMAR, R., ZYUBAN, V., AND TULLSEN, D. 2005. Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *ISCA*. 408–419.

LIU, Q., JIMENEZ, V., MORETO, M., ABELLA, J., CAZORLA, F., AND VALERO, M. 2013. Per-task energy accounting in computing systems. *IEEE Computer Architecture Letters (to appear). http://people.ac.upc.edu/jabella/camerareadyIEEECAL.pdf*.

MCCULLOUGH, J., AGARWAL, Y., CHANDRASHEKAR, J., KUPPUSWAMY, S., SNOEREN, A., AND GUPTA, R. 2011. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conference*. 12–12.

MICHALAKES, J., DUDHIA, J., GILL, D., HENDERSON, T., KLEMP, J., SKAMAROCK, W., AND WANG, W. The weather research and forecast model: software architecture and performance. In *11th Workshop on the Use of High Performance Computing in Meteorology, Reading*.

MORETO, M., CAZORLA, F., RAMIREZ, A., AND VALERO, M. 2008. Mlp-aware dynamic cache partitioning. In *HiPEAC*.

MURALIMANOHAR, N. AND BALASUBRAMONIAN, R. 2009. Cacti 6.0: A tool to understand large caches. *HP Tech Report HPL-2009-85*.

NAFFZIGER, S., STACKHOUSE, B., GRUTKOWSKI, T., JOSEPHSON, D., DESAI, J., ALON, E., AND HOROWITZ, M. 2005. The implementation of a 2-core multi-threaded itanium family processor. *IEEE Journal of Solid-State Circuits*, 182–183.

NAWATHE, U., HASSAN, M., WARRINER, L., YEN, K., GREENHILL, D., KUMAR, A., AND PARK, H. 2008. Implementation of an 8-core, 64-thread, power-efficient sparc server on a chip. *IEEE Journal of Solid-State Circuits, 43,* 1, 6–20.

NOKIA. 2012. Energy profiler.

PATHAK, A., HU, C., ZHANG, M., BAHL, P., AND WANG, W.-M. 2011. Fine-grained power modeling for smartphones using system call tracing. In *EuroSys*. 153–168.

RAGHAVENDRA, R., RANGANATHAN, P., TALWAR, V., WANG, Z., AND ZHU, X. 2008. No "power" struggles: coordinated multi-level power management for the data center. *ASPLOS*, 48–59.

SALMINEN, E., KANGAS, T., LAHTINEN, V., RIIHIMÄKI, J., KUUSILINNA, K., AND HÄMÄLÄINEN, T. 2007. Benchmarking mesh and hierarchical bus networks in system-on-chip context. *J. Syst. Archit. 53,* 8.

SHEN, K., SHRIRAMAN, A., DWARKADAS, S., ZHANG, X., AND CHEN, Z. 2013. Power containers: an os facility for fine-grained power and energy management on multicore servers. In *ASPLOS*.

SHERWOOD, T., PERELMAN, E., AND CALDER, B. 2001. Basic block distribution analysis to find periodic behavior and simulation points in applications. 3–14.

SINGHAL, R. 2008. Inside intel next generation nehalem microarchitecture. In *Intel Developer Forum*.

TULLSEN, D., EGGERS, S., AND LEVY, H. 1998. Simultaneous multithreading: maximizing on-chip parallelism. In *ISCA*. 533–544.

UDIPI, A., MURALIMANOHAR, N., AND BALASUBRAMONIAN, R. 2010. Towards scalable, energy-efficient, bus-based on-chip networks. In *HPCA*.

WESTE, N. AND ESHRAGHIAN, K. 1988. *Principles of CMOS VLSI Design. A Systems Perspective*. Addison-Wesley.

Received June 2013; revised September 2013; accepted October 2013